Description of STM32G4 HAL and low-layer drivers

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeG4 for STM32G4)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. HAL APIs are available for all peripherals.
  - Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS and USB.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks).

The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors.

The HAL drivers are feature-oriented instead of IP-oriented. For example, the timer APIs are split into several categories following the IP functions, such as basic timer, capture and pulse width modulation (PWM). The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Run-time detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities, and provide atomic operations that must be called by following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter or data pointers. All operations are performed by changing the content of the associated peripheral registers. Unlike the HAL, LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or a complex upper-level stack (such as USB).

The HAL and LL are complementary and cover a wide range of application requirements:

- The HAL offers high-level and feature-oriented APIs with a high-portability level. These hide the MCU and peripheral complexity from the end-user.
- The LL offers low-level APIs at register level, with better optimization but less portability. These require deep knowledge of the MCU and peripheral specifications.

The HAL- and LL-driver source code is developed in Strict ANSI-C, which makes it independent of the development tools. It is checked with the CodeSonar® static analysis tool. It is fully documented.

It is compliant with MISRA C®:2012 standard.

This user manual is structured as follows:

- Overview of HAL drivers
- Overview of low-layer drivers
- Cohabiting of HAL and LL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application

1 General information

The STM32CubeG4 MCU Package runs on STM32G4 32-bit microcontrollers based on the Arm® Cortex®-M processor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Acronyms and definitions

Table 1. Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex microcontroller software interface standard
COMP	Comparator
CORDIC	Trigonometric calculation unit
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
CSS	Clock security system
DAC	Digital to analog converter
DLYB	Delay block
DCMI	Digital camera interface
DFSDM	Digital filter sigma delta modulator
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
DSI	Display serial interface
DTS	Digital temperature sensor
ETH	Ethernet controller
EXTI	External interrupt/event controller
FDCAN	Flexible data-rate controller area network unit
FLASH	Flash memory
FMAC	Filtering mathematical calculation unit
FMC	Flexible memory controller
FW	Firewall
GFXMMU	Chrom-GRC
GPIO	General purpose I/Os
GTZC	Global TrustZone® controller
GTZC-MPCBB	GTZC block-based memory protection controller
GTZC-MPCWM	GTZC watermark memory protection controller
GTZC-TZIC	GTZC TrustZone illegal access controller
GTZC-TZSC	GTZC TrustZone security controller

Acronym	Definition
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB host controller driver
HRTIM	High-resolution timer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog
JPEG	Joint photographic experts group
LCD	Liquid crystal display controler
LTDC	LCD TFT Display Controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MDIOS	Management data input/output (MDIO) slave
MDMA	Master direct memory access
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND Flash memory
NOR	NOR Flash memory
NVIC	Nested vectored interrupt controller
OCTOSPI	Octo-SPI interface
OPAMP	Operational amplifier
OTFDEC	On-the-fly decryption engine
OTG-FS	USB on-the-go full-speed
PKA	Public key accelerator
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PSSI	Parallel synchronous slave interface
PWR	Power controller
QSPI	Quad-SPI Flash memory
RAMECC	RAM ECC monitoring
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface

Acronym	Definition
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial peripheral interface
SPDIFRX	SPDIF-RX Receiver interface
SRAM	SRAM external memory
SWPMI	Serial wire protocol master interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch sensing controller
TZ	Arm TrustZone-M
TZEN	TrustZone enable Flash user option bit
UART	Universal asynchronous receiver/transmitter
UCPD	USB Type-C® and Power Delivery interface
USART	Universal synchronous receiver/transmitter
VREFBUF	Voltage reference buffer
WWDG	Window watchdog
USB	Universal serial bus

3 Overview of HAL drivers

The HAL drivers are designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names.

The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (such as USART1 or USART2)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

3.1 HAL and user-application files

3.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
stm32g4xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32g4xx_hal_adc.c, stm32g4xx_hal_irda.c.*
stm32g4xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32g4xx_hal_adc.h,stm32g4xx_hal_irda.h.*
stm32g4xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32g4xx_hal_adc_ex.c,stm32g4xx_hal_flash_ex.c.*
stm32g4xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32g4xx_hal_adc_ex.h,stm32g4xx_hal_flash_ex.h.*
stm32g4xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32g4xx_hal.h	stm32g4xx_hal.c header file
stm32g4xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32g4xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32g4xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

3.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3. User-application files

File	Description
system_stm32g4xx.c	This file contains SystemInit() that is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM.
startup_stm32g4xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32g4xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly adapting the stack/heap size to fit the application requirements.
stm32g4xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.

File	Description
stm32g4xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32g4xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32g4xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

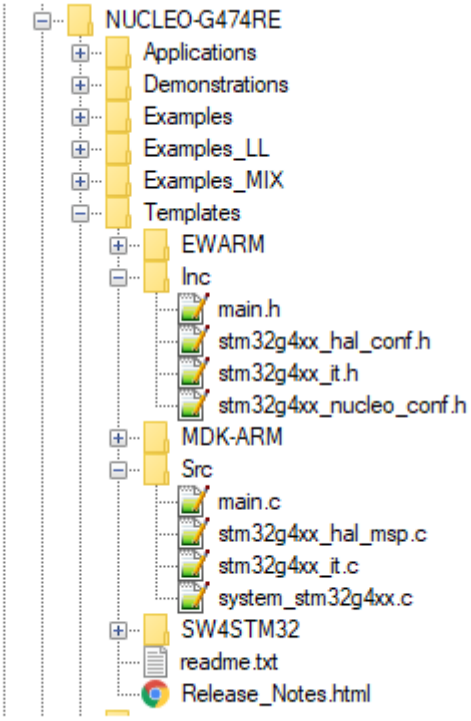
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized

Note: If an existing project is copied to another location, then include paths must be updated.

Figure 1. **Example of project template**



3.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

3.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that allows working with several IP instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi-instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.

  Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

Note:

1. *The multi-instance feature implies that all the APIs used in the application are reentrant and avoid using global variables because subroutines can fail to be reentrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:*
   - *Reentrant code does not hold any static (or global) non-constant data: reentrant functions can work with global data. For example, a reentrant interrupt service routine can grab a piece of hardware status to work with (for example serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.*
   - *Reentrant code does not modify its own code.*
2. *When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.*
3. *For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:*
   - *GPIO*
   - *SYSTICK*
   - *NVIC*
   - *PWR*
   - *RCC*
   - *FLASH*

3.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
{
  uint32_t BaudRate;                  /*!< This member configures the UART communication
baud rate.
  uint32_t WordLength;                /*!< Specifies the number of data bits transmitted or
received in a frame.
  uint32_t StopBits;                  /*!< Specifies the number of stop bits transmitted.
   uint32_t Parity;                    /*!< Specifies the parity mode.
  uint32_t Mode;                      /*!< Specifies whether the Receive or Transmit mode is
enabled or disabled.
  uint32_t HwFlowCtl;                 /*!< Specifies whether the hardware flow control mode
is enabled
                                           or disabled.
  uint32_t OverSampling;              /*!< Specifies whether the Over sampling 8 is enabled
or disabled, to achieve higher speed (up to f_PCLK/8).
                                           This parameter can be a value of @ref
UART_Over_Sampling. */
  uint32_t OneBitSampling;            /*!< Specifies whether a single sample or three
samples' majority vote is selected.
  uint32_t ClockPrescaler;            /*!< Specifies the prescaler value used to divide the
UART clock source.
                                           This parameter can be a value of @ref
UART_ClockPrescaler. */
} UART_InitTypeDef;
```

Note: The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)
```

3.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

3.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:**

  This set of API is divided into two sub-categories :

  – **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc,uint32_t
SingleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff);
```

  – **Feature/Device specific APIs**: these APIs are implemented in the extension file and delimited by specific define statements relative to CMSIS bits or features and depending on device part number.

```
#if defined(PWR_SHDW_SUPPORT)
void HAL_PWREx_EnterSHUTDOWNMode(void);
#endif
```

Note: The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4. API classification

	Generic file	Extension file
Common APIs	X	X
Family specific APIs	-	X
Device specific APIs	-	X

Note: Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.

Note: The IRQ handlers are used for common and family specific processes.

3.4 Devices supported by HAL drivers

Table 5. **List of devices supported by HAL drivers**

IP/Module	STM32G431xx	STM32G441xx	STM32G471xx	STM32G473xx	STM32G483xx	STM32G474xx	STM32G484xx	STM32GBK1CB	STM32G491xx	STM32G4A1xx
stm32g4xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_cordic.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_cryp.c	No	Yes	No	No	Yes	No	Yes	No	No	Yes
stm32g4xx_hal_cryp_ex.c	No	Yes	No	No	Yes	No	Yes	No	No	Yes
stm32g4xx_hal_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_exti.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_fdcan.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_flash_ramfunc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_fmac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_hrtim.c	No	No	No	No	No	Yes	Yes	No	No	No
stm32g4xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_i2s.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_msp_template.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_nand.c	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes
stm32g4xx_hal_nor.c	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes
stm32g4xx_hal_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_opamp_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_pcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32G431xx	STM32G441xx	STM32G471xx	STM32G473xx	STM32G483xx	STM32G474xx	STM32G484xx	STM32GBK1CB	STM32G491xx	STM32G4A1xx
stm32g4xx_hal_pcd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_qspi.c	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_sai.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_sai_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_sram.c	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes
stm32g4xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_timebase_tim_template.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_usart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_cordic.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_crs.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_exti.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_fmac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32G431xx	STM32G441xx	STM32G471xx	STM32G473xx	STM32G483xx	STM32G474xx	STM32G484xx	STM32GBK1CB	STM32G491xx	STM32G4A1xx
stm32g4xx_ll_fmc.c	No	No	No	Yes	Yes	Yes	Yes	No	No	No
stm32g4xx_ll_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_hrtim.c	No	No	No	No	No	Yes	Yes	No	No	No
stm32g4xx_ll_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_lpuart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_ucpd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_usb.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32g4xx_ll_utils.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

3.5 HAL driver rules

3.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6. HAL API naming rules

	Generic	Family specific	Device specific
File names	*stm32g4xx_hal_ppp (c/h)*	*stm32g4xx_hal_ppp_ex (c/h)*	*stm32g4xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32G4 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in stm32g4xxx.h header file:

  stm32g4xxx.h corresponds to stm32g431xx.h, stm32g441xx.h, stm32g471xx.h, stm32g473xx.h, stm32g483xx.h, stm32g474xx.h, stm32g484xx.h, stm32gbk1cb.h and stm32g491xx.h and stm32g4a1xx.h.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (for example HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (for example ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (for example ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (for example HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named HAL_PPP_DeInit (for example HAL_TIM_DeInit()).
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA ()*.
- The **Feature** prefix should refer to the new feature.

3.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH.

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below:

Note: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7. Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the stm32g4xx_hal_cortex.c file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example : STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)".
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
 HAL_PPP_Init(PPP_HandleTypeDef)
if(hppp == NULL)
 {
 return HAL_ERROR;
 }
```

- The macros defined below are used:
  - Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x) : -(x))
```

  - Pseudo-code macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32g4xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8. Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Example: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Example: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Example: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

3.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (such as PWM, OC and IC).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()*function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in run time the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 9. HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used

Function group	Common API name	Description
IO operation group	*HAL_ADC_Stop ()*	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in run time the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in run time the error that occurred during IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32g4xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32g4xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10. HAL extension APIs

Function group	Common API name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration
HAL_ADCEx_Calibration_GetValue()	This function is used to get the ADC calibration factor

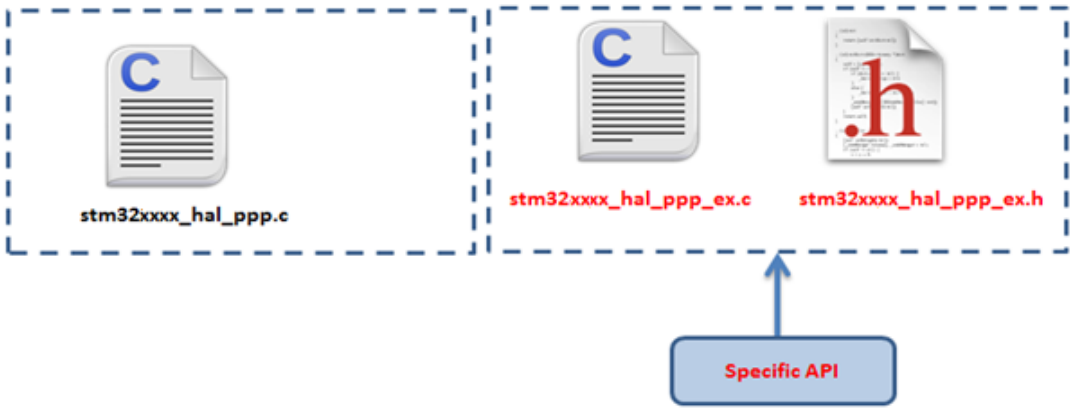
3.7.2 HAL extension model cases

The specific IP features can be handled by the HAL drivers in five different ways. They are described below.

Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32g4xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

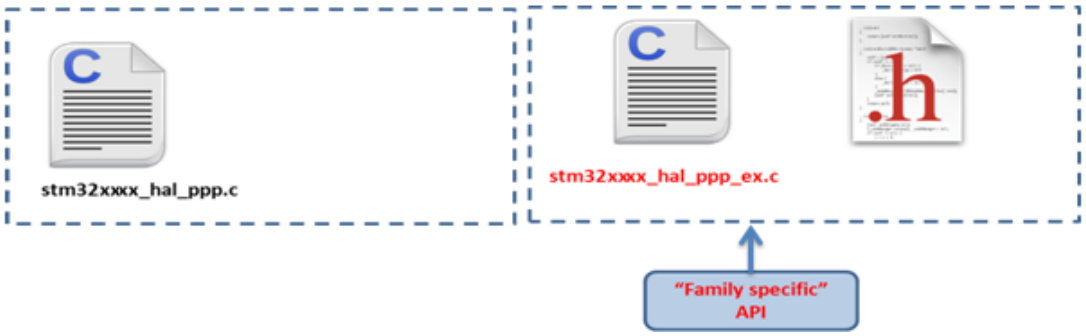
Figure 2. **Adding device-specific functions**



Adding a family-specific function

In this case, the API is added in the extension driver C file and named HAL_PPPEx_Function ().

Figure 3. **Adding family-specific functions**

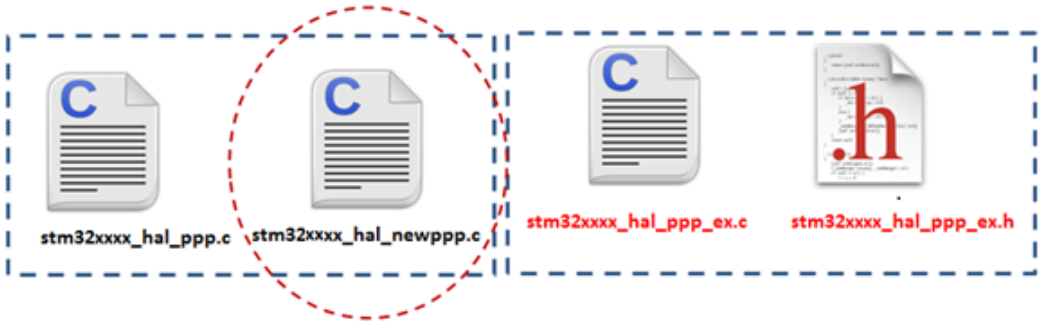


Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32g4xx_hal_newppp.c. However the inclusion of this file is selected in the stm32g4xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. **Adding new peripherals**

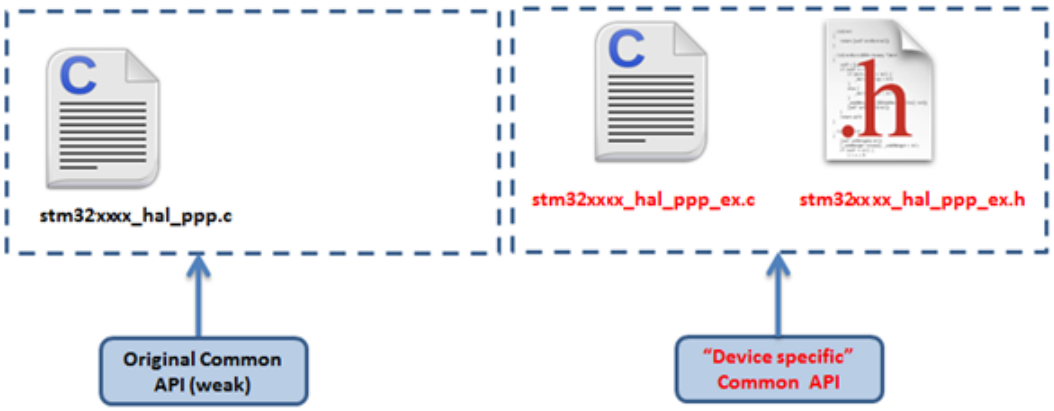


Example: stm32g4xx_hal_adc.c/h

Updating existing common APIs

In this case, the routines are defined with the same names in the stm32g4xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler overwrites the original routine by the new defined function.

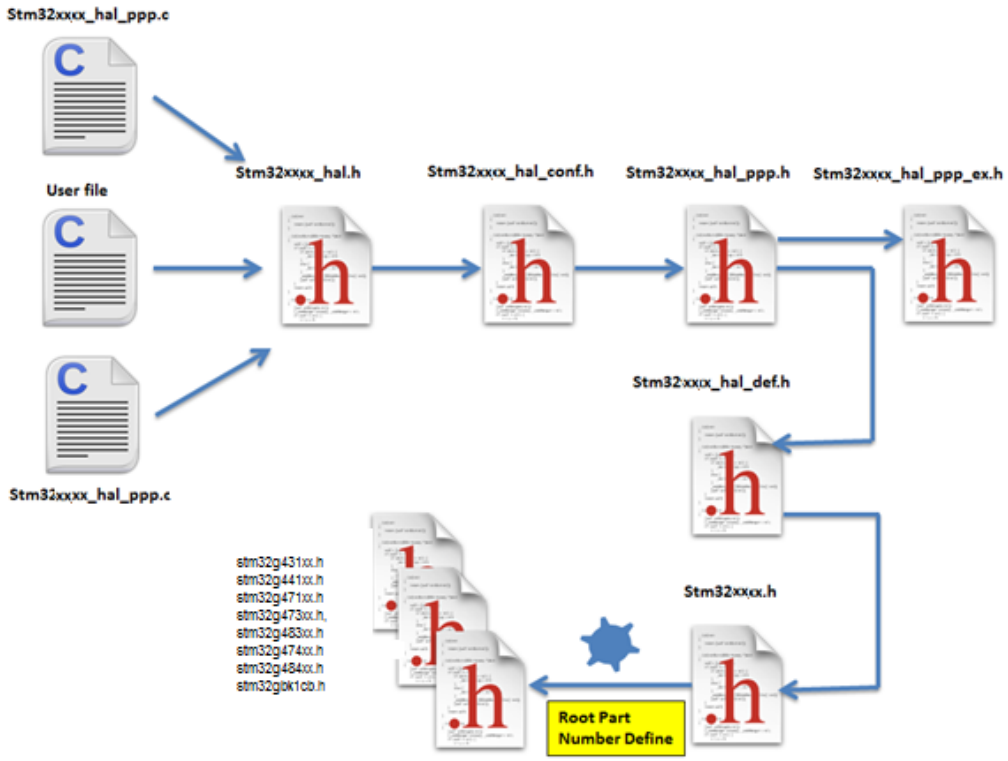
Figure 5. Updating existing APIs



3.8 File inclusion model

The header of the common HAL driver file (stm32g4xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6. File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```
/**********************************************************************************
  * @file    stm32g4xx_hal_conf.h
  * @author  MCD Application Team
  * @brief   HAL configuration template file.
  *          This file should be copied to the application folder and renamed
  *          to stm32g4xx_hal_conf.h.
  **********************************************************************************
(…)
#define HAL_USART_MODULE_ENABLED
#define HAL_IRDA_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
(…)
```

3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32g4xx_hal_def.h.*The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status**

  The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
Typedef enum
{
HAL_OK = 0x00,
HAL_ERROR = 0x01,
HAL_BUSY = 0x02,
HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked**

  The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

  In addition to common resources, the stm32g4xx_hal_def.h file calls the stm32g4xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:

  – Declarations of peripheral registers and bits definition.
  – Macros to access peripheral registers hardware (such as Write register or Read register).

- **Common macros**

  – Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  – Macro linking a PPP peripheral to a DMA structure pointer:

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
  (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
  (__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.10 HAL configuration

The configuration file, stm32g4xx_hal_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable or modify some options by uncommenting, commenting or modifying the values of the related define statements as described in the table below:

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	100
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 Hz
LSI_VALUE	Defines the default value of the Low-speed internal oscillator (LSI) expressed in Hz.	32000 Hz
LSE_VALUE	Defines the value of the external oscillator (LSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz

Configuration item	Description	Default Value
LSE_STARTUP_TIMEOUT	Timeout for LSE start-up, expressed in ms	5000
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	FALSE
INSTRUCTION_CACHE_ENABLE	Enables I-cache feature	TRUE
DATA_CACHE_ENABLE	Enables D-cache feature	TRUE

Note: The stm32g4xx_hal_conf_template.h file is located in the HAL drivers Inc folder. It should be copied to the user folder, renamed and modified as described above.

Note: By default, the values defined in the stm32g4xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

3.11 HAL system peripheral handling

This chapter gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

3.11.1 Clock

Two main functions can be used to configure the system clock:

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - selects the system clock source
  - configures clock dividers
  - configures the number of Flash memory wait states
  - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC, USB). In this case, the clock configuration is performed by an extended API defined in stm32g4xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-initialization function that returns clock configuration to reset state
- Get clock functions that allow retreiving various clock configurations (such as system clock, HCLK, PCLK1 or PCLK2)
- MCO and CSS configuration functions

A set of macros are defined in stm32g4xx_hal_rcc.h and stm32g4xx_hal_rcc_ex.h. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __HAL_PPP_CLK_ENABLE/__HAL_PPP_CLK_DISABLE to enable/disable the peripheral clock
- __HAL_PPP_FORCE_RESET/__HAL_PPP_RELEASE_RESET to force/release peripheral reset
- __HAL_PPP_CLK_SLEEP_ENABLE/__HAL_PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during Sleep mode.
- __HAL_PPP_IS_CLK_ENABLED/__HAL_PPP_IS_CLK_DISABLED to query about the enabled/disabled status of the peripheral clock.
- __HAL_PPP_IS_CLK_SLEEP_ENABLED/__HAL_PPP_IS_CLK_SLEEP_DISABLED to query about the enabled/disabled status of the peripheral clock during Sleep mode.

3.11.2 GPIOs

GPIO HAL APIs are the following:

- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
- HAL_GPIO_TogglePin ().

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32g4xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 12. Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input floating – GPIO_MODE_OUTPUT_PP : Output push-pull – GPIO_MODE_OUTPUT_OD : Output open drain – GPIO_MODE_AF_PP : Alternate function push-pull – GPIO_MODE_AF_OD : Alternate function open drain – GPIO_MODE_ANALOG : Analog mode • External Interrupt mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs:

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

3.11.3 Cortex® NVIC and SysTick timer

The Cortex® HAL driver, stm32g4xx_hal_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL_NVIC_SetPriority()/ HAL_NVIC_SetPriorityGrouping()
- HAL_NVIC_GetPriority() / HAL_NVIC_GetPriorityGrouping()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_NVIC_GetActive(IRQn)
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()

3.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
  - HAL_PWREx_PVD_PVM_IRQHandler
  - HAL_PWR_PVDCallback()
- Wakeup pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTOPMode() (kept for compatibility with other family but identical to HAL_PWREx_EnterSTOP0Mode() or HAL_PWREx_EnterSTOP1Mode() (see hereafter)
  - HAL_PWR_EnterSTANDBYMode()
- STM32G4 low-power management features:
  - HAL_PWREx_EnterSTOP0Mode()
  - HAL_PWREx_EnterSTOP1Mode()
  - HAL_PWREx_EnterSTOP2Mode()
  - HAL_PWREx_EnterSHUTDOWNMode()

3.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral, that are handled through EXTI HAL APIs. In addition, each peripheral HAL driver implements the associated EXTI configuration and function as macros in its header file.

The first 16 EXTI lines connected to the GPIOs are managed within the GPIO driver. The GPIO_InitTypeDef structure allows configuring an I/O as external interrupt or external event.

The EXTI lines connected internally to the PVD, RTC, USB, and Ethernet are configured within the HAL drivers of these peripheral through the macros given in the table below.

The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13. Description of EXTI configuration macros

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_GENERATE_SWIT()	Generates a software interrupt for a given EXTI line. Example: __HAL_PWR_PVD_EXTI_ GENERATE_SWIT ()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()	Enable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()	Disable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()	Configure an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()	Disable an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Rising/Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32g4xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

3.11.6 DMA

The DMA HAL driver allows enabling and configuring the peripheral to be connected to the DMA Channels (except for internal SRAM/FLASH memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given channel, HAL_DMA_Init() API allows programming the required configuration through the following parameters:

- Transfer direction
- Source and destination data formats
- Circular, Normal control mode
- Channel priority level
- Source and destination Increment mode

Two operating modes are available:

- Polling mode I/O operation
  1. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
  2. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
  1. Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
  2. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
  3. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been confgured. In this case the DMA interrupt is configured.
  4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
  5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel.
- __HAL_DMA_DISABLE: disables the specified DMA channel.
- __HAL_DMA_GET_FLAG: gets the DMA channel pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA channel pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA channel interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA channel interrupts.
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA channel interrupt has been enabled or not.

Note: When a peripheral is used in DMA mode, the DMA initialization should be done in the HAL_PPP_MspInit() callback. In addition, the user application should associate the DMA handle to the PPP handle (refer to section "HAL IO operation functions").

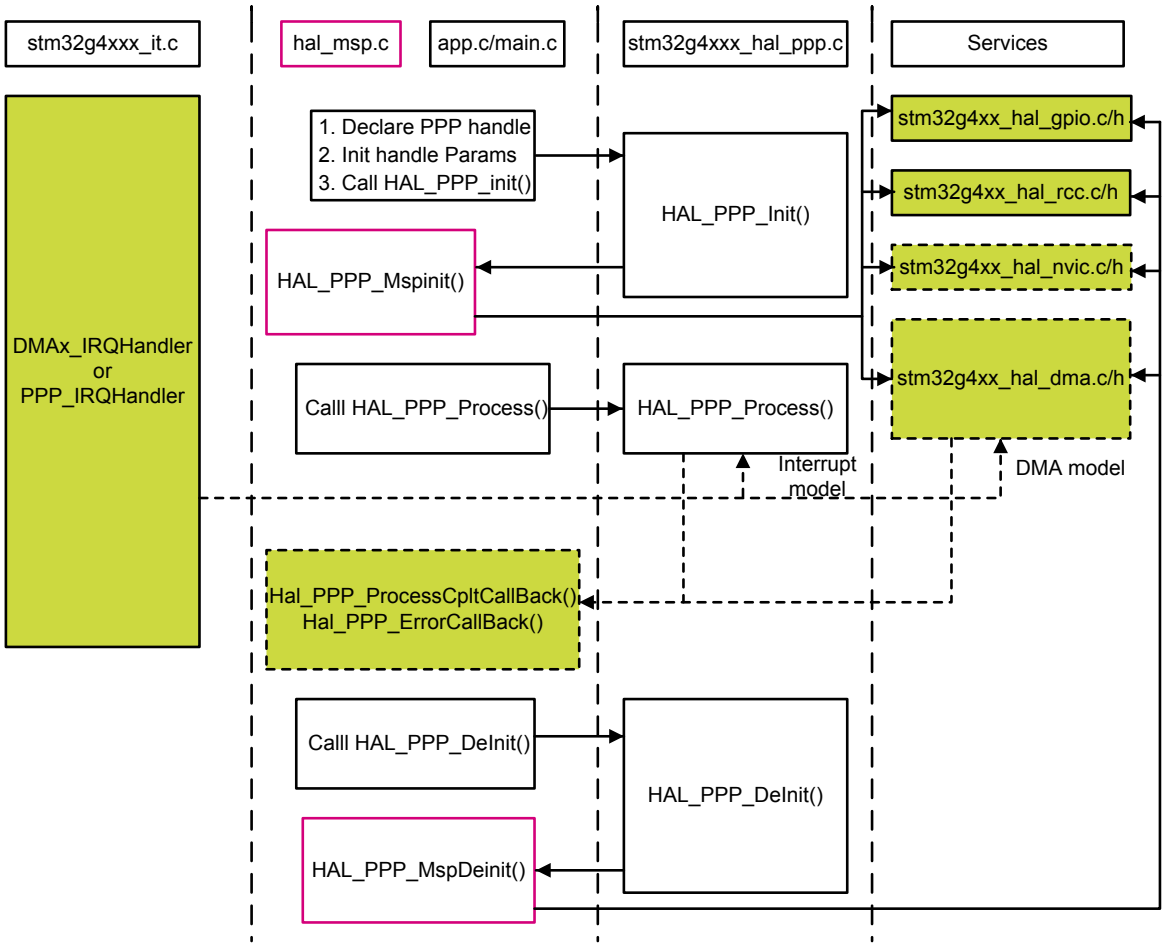
Note: DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

3.12 How to use HAL drivers

3.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 7. **HAL driver model**



Note: The functions implemented in the HAL driver are shown in green, the functions called from interrupt handlers in dotted lines, and the msp functions implemented in the user application in red. Non-dotted lines represent the interactions between the user application functions.

Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

3.12.2 HAL initialization

3.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32g4xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in the HAL drivers.
- HAL_DeInit()
  - resets all peripherals
  - calls function HAL_MspDeInit() which is a user callback function to do system level De-Initializations.

- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.

  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR is blocked.

3.12.2.2 *System clock initialization*

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code.

Please find below the typical Clock configuration sequence to reach the maximum 80 MHz clock frequency based on the HSE clock:

```
void SystemClock_Config(void)
{
RCC_ClkInitTypeDef clkinitstruct = {0};
RCC_OscInitTypeDef oscinitstruct = {0};
/* Configure PLLs------------------------------------------------------*/
/* PLL configuration: PLLCLK = (HSE/PLLM * PLLN) / PLLR = (16/1 * 20) / 2 = 80
MHz*/
/* Enable HSE Oscillator and activate PLL with HSE as source */
oscinitstruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
oscinitstruct.HSEState = RCC_HSE_ON;
oscinitstruct.PLL.PLLState = RCC_PLL_ON;
oscinitstruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
oscinitstruct.PLL.PLLM = 1;
oscinitstruct.PLL.PLLN = 20;
oscinitstruct.PLL.PLLR = 2;
oscinitstruct.PLL.PLLL = 7;
oscinitstruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
Overview of HAL drivers UM1884
64/2232 DocID027704 Rev 7
{
/* Initialization Error */
while(1);
}
/* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
clocks dividers */
clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK I RCC_CLOCKTYPE_HCLK I
RCC_CLOCKTYPE_PCLK1 I RCC_CLOCKTYPE_PCLK2);
clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV1;
if
(HAL_RCC_ClockConfig(&clkinitstruct,FLASH_LATENCY_4)!= HAL_OK)
{
/* Initialization Error */
while(1);
}
}
```

3.12.2.3 *HAL MSP initialization process*

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit().*

The MspInit callback performs the low level initialization related to the different additional hardware resources: RCC, GPIO, NVIC and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and de-initialization:

```
/**
* @brief Initializes the PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}
/**
* @brief DeInitializes PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low level initialization code or omit them and use his own initialization routine.

The HAL MSP callback is implemented inside the *stm32g4xx_hal_msp.c* file in the user folders. An *stm32g4xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32g4xx_hal_msp.c file contains the following functions:

Table 14. MSP functions

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP De-Initialization in the *Hal_MspDeInit()*. In this case the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be de-initialized in run time and the low level resources of a given peripheral need to be released and used by another peripheral, HAL_PPP_MspDeInit() and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

3.12.3 HAL I/O operation process

The HAL functions with internal data processing like transmit, receive, write and read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

3.12.3.1 Polling mode

In Polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical Polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(…)
return HAL_OK; }
```

3.12.3.2 *Interrupt mode*

In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In Interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launches the process
- *HAL_PPP_IRQHandler()*: global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: callback relative to the process Error.

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32g4xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

*stm32g4xx_it.c*file:

```
extern UART_HandleTypeDef UartHandle;
void USART1_IRQHandler(void)
{
HAL_UART_IRQHandler(&UartHandle);
}
```

3.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32g4xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
__HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver that means, the user can declare it again in the application code. The function in the driver should not be modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32g4xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

3.12.4 Timeout and error management

3.12.4.1 Timeout management

The timeout is often used for the APIs that operate in Polling mode. It defines the delay during which a blocking process should wait till an error is returned. An example is provided below:

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel,
uint32_t Timeout)
```

The timeout possible values are the following:

Table 15. Timeout values

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) (1)	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

1. *HAL_MAX_DELAY is defined in the stm32g4xx_hal_def.h as 0xFFFFFFFF*

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef)
{
(…)
timeout = HAL_GetTick() + LOCAL_PROCESS_TIMEOUT;
(…)
while(ProcessOngoing)
{
(…)
if(HAL_GetTick() ≥ timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return HAL_PPP_STATE_TIMEOUT;
}
}
(…)
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(…)
timeout = HAL_GetTick() + Timeout;
(…)
while(ProcessOngoing)
{
(…)
if(Timeout != HAL_MAX_DELAY)
{
if(HAL_GetTick() ≥ timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(…)
}
```

3.12.4.2 *Error management*

The HAL drivers implement a check on the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system may crash or go into an undefined state. These critical parameters are checked before being used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32
Size)
{
if ((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL_StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{
if (hppp == NULL) //the handle should be already allocated
{
return HAL_ERROR;
}
}
```

- Timeout error: the following statement is used when a timeout error occurs:

```
while (Process ongoing)
{
timeout = HAL_GetTick() + Timeout; while (data processing is running)
{
if(timeout) { return HAL_TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to allow retrieving the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY;  /* Set the peripheral ready */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
ErrorCode = HAL_PPP_GetError (hppp); /* retreive error code */
}
```

3.12.4.3 *Run-time checking*

The HAL implements run-time failure detection by checking the input values of all HAL driver functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

To enable the run-time checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in *stm32g4xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
(..) /* Check the parameters */
assert_param(IS_UART_INSTANCE(huart->Instance));
assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
assert_param(IS_UART_PARITY(huart->Init.Parity));
assert_param(IS_UART_MODE(huart->Init.Mode));
assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
(..)
```

```
/** @defgroup UART_Word_Length *
@{
*/
#define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
#define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
#define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
\ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the assert_param macro is false, the*assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in stm32g4xx_hal_conf.h:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr: If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions ------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The *assert_failed* function is implemented in the main.c file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* Infinite loop */
while (1)
{
}
}
```

Note: ***Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.***

4 Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as USB).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features.

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

4.1 Low-layer files

The low-layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some System and Cortex related features.

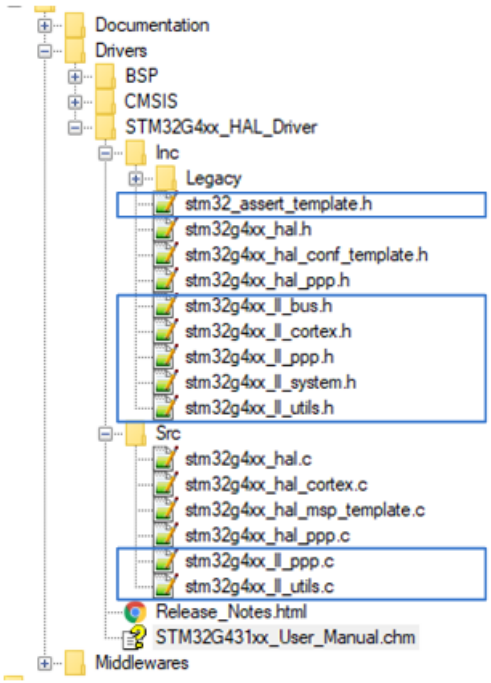
Table 16. LL driver files

File	Description
stm32g4xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32g4xx_ll_ppp.h/.c	stm32g4xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32g4xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32g4xx_ll_ppp.h file.
stm32g4xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (such as LL_SYSTICK_xxxxx and LL_LPM_xxxxx "Low Power Mode")
stm32g4xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
stm32g4xx_ll_system.h	System related operations. Example: LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx and LL_VREFBUF_xxx
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

Note: There is no configuration file for the LL drivers.

The low-layer files are located in the same HAL driver folder.

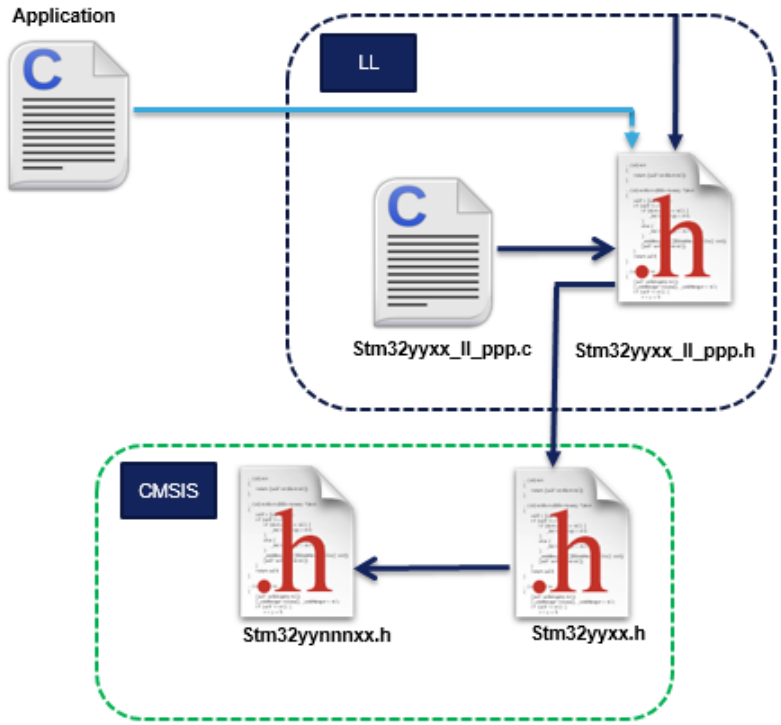
Figure 8. Low-layer driver folders



In general, low-layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 9. **Low-layer driver CMSIS files**



Application files have to include only the used low-layer driver header files.

4.2 Overview of low-layer APIs and naming rules

4.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in stm32g4xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral de-initialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 17. Common peripheral initialization functions

Functions	Return Type	Parameters	Description
LL_PPP_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to Table 18. Optional peripheral initialization functions).

Table 18. Optional peripheral initialization functions

Functions	Return Type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)
LL_PPP_CommonInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each *PPP_CommonInitStruct* member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	Initializes the peripheral clock configuration in synchronous mode.

Functions	Return Type	Parameters	Examples
		• *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

4.2.1.1 *Run-time checking*

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL driver functions. For more details please refer to Section 3.12.4.3 Run-time checking.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use run-time checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when run-time checking is enabled.
2. Include stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.

Note: Run-time checking is not available for LL inline functions.

4.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending to the action category:

- **Specific Interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 19. Specific Interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 20. Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

Note: BITNAME refers to the peripheral register bit name as described in the product line reference manual.

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 21. Peripheral clock activation/deactivation management

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA| LL_AHB2_GRP1_PERIPH_GPIOB)* • *LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

Note: 'x' corresponds to the group index and refers to the index of the modified register on a given bus. 'bus' corresponds to the bus name.

- ***Peripheral activation/deactivation management*** : Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 22. Peripheral activation/deactivation management

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management*** : Set/get a peripheral configuration settings

Table 23. Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management*** : Write/read the content of a register/retrun DMA relative register address

Table 24. Peripheral register management

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

Note: The Propriety is a variable used to identify the DMA transfer direction or the data register type.

5 Cohabiting of HAL and LL

The low-ayer APIs are designed to be used in standalone mode or combined with the HAL. They cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the low-layer APIs might overwrite some registers which content is mirrored in the HAL handles.

5.1 Low-layer driver used in Standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including stm32g4xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case the HAL drivers associated to the used peripheral can be removed from the workspace. However the STM32CubeG4 framework should be used in the same way as in the HAL drivers case which means that System file, startup file and CMSIS should always be used.

Note: When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

5.2 Mixed use of low-layer APIs and HAL drivers

In this case the low-layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of low-layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the low-layer services can be used together for the same peripheral instance.
- The low-layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, flash and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within stm32g4 firmware package (refer to Examples_MIX projects).

Note:

1. *When the HAL Init/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.*
2. *When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.*
3. *When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.*

6 HAL System Driver

6.1 HAL Firmware driver API description

The following section lists the various functions of the HAL library.

6.1.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

6.1.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize the Flash interface the NVIC allocation and initial time base clock configuration.
- De-Initialize common part of the HAL.
- Configure the time base source to have 1ms time base with a dedicated Tick interrupt priority.
  - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
  - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
  - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  - functions affecting time base configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- ***HAL_Init***
- ***HAL_DeInit***
- ***HAL_MspInit***
- ***HAL_MspDeInit***
- ***HAL_InitTick***

6.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier

This section contains the following APIs:

- ***HAL_IncTick***
- ***HAL_GetTick***
- ***HAL_GetTickPrio***
- ***HAL_SetTickFreq***

- *HAL_GetTickFreq*
- *HAL_Delay*
- *HAL_SuspendTick*
- *HAL_ResumeTick*
- *HAL_GetHalVersion*
- *HAL_GetREVID*
- *HAL_GetDEVID*

6.1.4 HAL Debug functions

This section provides functions allowing to:

- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP0/STOP1/STOP2 modes
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- *HAL_DBGMCU_EnableDBGSleepMode*
- *HAL_DBGMCU_DisableDBGSleepMode*
- *HAL_DBGMCU_EnableDBGStopMode*
- *HAL_DBGMCU_DisableDBGStopMode*
- *HAL_DBGMCU_EnableDBGStandbyMode*
- *HAL_DBGMCU_DisableDBGStandbyMode*

6.1.5 HAL SYSCFG configuration functions

This section provides functions allowing to:

- Start a hardware CCMSRAM erase operation
- Enable/Disable the Internal FLASH Bank Swapping
- Configure the Voltage reference buffer
- Enable/Disable the Voltage reference buffer
- Enable/Disable the I/O analog switch voltage booster

This section contains the following APIs:

- *HAL_SYSCFG_CCMSRAMErase*
- *HAL_SYSCFG_EnableMemorySwappingBank*
- *HAL_SYSCFG_DisableMemorySwappingBank*
- *HAL_SYSCFG_VREFBUF_VoltageScalingConfig*
- *HAL_SYSCFG_VREFBUF_HighImpedanceConfig*
- *HAL_SYSCFG_VREFBUF_TrimmingConfig*
- *HAL_SYSCFG_EnableVREFBUF*
- *HAL_SYSCFG_DisableVREFBUF*
- *HAL_SYSCFG_EnableIOSwitchBooster*
- *HAL_SYSCFG_DisableIOSwitchBooster*
- *HAL_SYSCFG_EnableIOSwitchVDD*
- *HAL_SYSCFG_DisableIOSwitchVDD*
- *HAL_SYSCFG_CCMSRAM_WriteProtectionEnable*

6.1.6 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

This function is used to configure the Flash prefetch, the Instruction and Data caches, the time base source, NVIC and any required global low level hardware by calling the HAL_MspInit() callback function to be optionally defined in user file stm32g4xx_hal_msp.c.

Return values

- **HAL:** status

Notes

- HAL_Init() function is called at the beginning of program after reset and before the clock configuration.
- In the default implementation the System Timer (Systick) is used as source of time base. The Systick configuration is based on HSI clock, as HSI is the clock used after a system Reset and the NVIC configuration is set to Priority group 4. Once done, time base tick starts incrementing: the tick variable counter is incremented each 1ms in the SysTick_Handler() interrupt handler.

HAL_DeInit

Function name

HAL_StatusTypeDef HAL_DeInit (void)

Function description

This function de-initializes common part of the HAL and stops the source of time base.

Return values

- **HAL:** status

Notes

- This function is optional.

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initialize the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base: The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

- This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().
- In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name

void HAL_IncTick (void)

Function description

This function is called to increment a global variable "uwTick" used as application time base.

Return values

- **None:**

Notes

- In the default implementation, this variable is incremented each 1ms in SysTick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name

uint32_t HAL_GetTick (void)

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTickPrio

Function name

uint32_t HAL_GetTickPrio (void)

Function description

This function returns a tick priority.

Return values

- **tick:** priority

HAL_SetTickFreq

Function name

HAL_StatusTypeDef HAL_SetTickFreq (uint32_t Freq)

Function description

Set new tick Freq.

Return values

- **status:**

HAL_GetTickFreq

Function name

uint32_t HAL_GetTickFreq (void)

Function description

Returns tick frequency.

Return values

- **tick:** period in Hz

HAL_SuspendTick

Function name

void HAL_SuspendTick (void)

Function description

Suspends Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name

void HAL_ResumeTick (void)

Function description

Resume Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name

uint32_t HAL_GetHalVersion (void)

Function description

Returns the HAL revision.

Return values

- **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name

uint32_t HAL_GetREVID (void)

Function description

Returns the device revision identifier.

Return values

- **Device:** revision identifier

HAL_GetDEVID

Function name

uint32_t HAL_GetDEVID (void)

Function description

Returns the device identifier.

Return values

- **Device:** identifier

HAL_DBGMCU_EnableDBGSleepMode

Function name

void HAL_DBGMCU_EnableDBGSleepMode (void)

Function description

Enable the Debug Module during SLEEP mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGSleepMode

Function name

void HAL_DBGMCU_DisableDBGSleepMode (void)

Function description

Disable the Debug Module during SLEEP mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStopMode

Function name

void HAL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during STOP0/STOP1/STOP2 modes.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name

void HAL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP0/STOP1/STOP2 modes.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name

void HAL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name

void HAL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_SYSCFG_CCMSRAMErase

Function name

void HAL_SYSCFG_CCMSRAMErase (void)

Function description

Start a hardware CCMSRAM erase operation.

Return values

- **None:**

Notes

- As long as CCMSRAM is not erased the CCMER bit will be set. This bit is automatically reset at the end of the CCMSRAM erase operation.

HAL_SYSCFG_EnableMemorySwappingBank

Function name

void HAL_SYSCFG_EnableMemorySwappingBank (void)

Function description

Enable the Internal FLASH Bank Swapping.

Return values

- **None:**

Notes

- This function can be used only for STM32G4xx devices.
- Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08040000 (and aliased at 0x00040000)

HAL_SYSCFG_DisableMemorySwappingBank

Function name

void HAL_SYSCFG_DisableMemorySwappingBank (void)

Function description

Disable the Internal FLASH Bank Swapping.

Return values

- **None:**

Notes

- This function can be used only for STM32G4xx devices.
- The default state : Flash Bank1 mapped at 0x08000000 (and aliased @0x0000 0000) and Flash Bank2 mapped at 0x08040000 (and aliased at 0x00040000)

HAL_SYSCFG_VREFBUF_VoltageScalingConfig

Function name

void HAL_SYSCFG_VREFBUF_VoltageScalingConfig (uint32_t VoltageScaling)

Function description

Configure the internal voltage reference buffer voltage scale.

Parameters

- **VoltageScaling:** specifies the output voltage to achieve This parameter can be one of the following values:
  - SYSCFG_VREFBUF_VOLTAGE_SCALE0: VREFBUF_OUT around 2.048 V. This requires VDDA equal to or higher than 2.4 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE1: VREFBUF_OUT around 2.5 V. This requires VDDA equal to or higher than 2.8 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE2: VREFBUF_OUT around 2.9 V. This requires VDDA equal to or higher than 3.15 V.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_HighImpedanceConfig

Function name

void HAL_SYSCFG_VREFBUF_HighImpedanceConfig (uint32_t Mode)

Function description

Configure the internal voltage reference buffer high impedance mode.

Parameters

- **Mode:** specifies the high impedance mode This parameter can be one of the following values:
  - SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE: VREF+ pin is internally connect to VREFINT output.
  - SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE: VREF+ pin is high impedance.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_TrimmingConfig

Function name

void HAL_SYSCFG_VREFBUF_TrimmingConfig (uint32_t TrimmingValue)

Function description

Tune the Internal Voltage Reference buffer (VREFBUF).

Parameters

- **TrimmingValue:** specifies trimming code for VREFBUF calibration This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x3F

Return values

- **None:**

HAL_SYSCFG_EnableVREFBUF

Function name

HAL_StatusTypeDef HAL_SYSCFG_EnableVREFBUF (void)

Function description

Enable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **HAL_OK/HAL_TIMEOUT:**

HAL_SYSCFG_DisableVREFBUF

Function name

void HAL_SYSCFG_DisableVREFBUF (void)

Function description

Disable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_SYSCFG_EnableIOSwitchBooster

Function name

void HAL_SYSCFG_EnableIOSwitchBooster (void)

Function description

Enable the I/O analog switch voltage booster.

Return values

- **None:**

HAL_SYSCFG_DisableIOSwitchBooster

Function name

void HAL_SYSCFG_DisableIOSwitchBooster (void)

Function description

Disable the I/O analog switch voltage booster.

Return values

- **None:**

HAL_SYSCFG_EnableIOSwitchVDD

Function name

void HAL_SYSCFG_EnableIOSwitchVDD (void)

Function description

Enable the I/O analog switch voltage by VDD.

Return values

- **None:**

HAL_SYSCFG_DisableIOSwitchVDD

Function name

void HAL_SYSCFG_DisableIOSwitchVDD (void)

Function description

Disable the I/O analog switch voltage by VDD.

Return values

- **None:**

HAL_SYSCFG_CCMSRAM_WriteProtectionEnable

Function name

void HAL_SYSCFG_CCMSRAM_WriteProtectionEnable (uint32_t Page)

Function description

CCMSRAM page write protection enable.

Parameters

- **Page:** This parameter is a long 32bit value and can be a value of CCM Write protection

Return values

- **None:**

Notes

- write protection can only be disabled by a system reset

6.2 HAL Firmware driver defines

The following section lists the various define and macros of the module.

6.2.1 HAL

HAL

DBGMCU Exported Macros

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM4

__HAL_DBGMCU_UNFREEZE_TIM4

__HAL_DBGMCU_FREEZE_TIM5

__HAL_DBGMCU_UNFREEZE_TIM5

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_RTC

__HAL_DBGMCU_UNFREEZE_RTC

__HAL_DBGMCU_FREEZE_WWDG

__HAL_DBGMCU_UNFREEZE_WWDG

__HAL_DBGMCU_FREEZE_IWDG

__HAL_DBGMCU_UNFREEZE_IWDG

__HAL_DBGMCU_FREEZE_I2C1_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT

__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT

__HAL_DBGMCU_FREEZE_I2C3_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT

__HAL_DBGMCU_FREEZE_LPTIM1

__HAL_DBGMCU_UNFREEZE_LPTIM1

__HAL_DBGMCU_FREEZE_I2C4_TIMEOUT

__HAL_DBGMCU_UNFREEZE_I2C4_TIMEOUT

__HAL_DBGMCU_FREEZE_TIM1

__HAL_DBGMCU_UNFREEZE_TIM1

__HAL_DBGMCU_FREEZE_TIM8

__HAL_DBGMCU_UNFREEZE_TIM8

__HAL_DBGMCU_FREEZE_TIM15

__HAL_DBGMCU_UNFREEZE_TIM15

__HAL_DBGMCU_FREEZE_TIM16

__HAL_DBGMCU_UNFREEZE_TIM16

__HAL_DBGMCU_FREEZE_TIM17

__HAL_DBGMCU_UNFREEZE_TIM17

__HAL_DBGMCU_FREEZE_TIM20

__HAL_DBGMCU_UNFREEZE_TIM20

__HAL_DBGMCU_FREEZE_HRTIM1

__HAL_DBGMCU_UNFREEZE_HRTIM1

HAL FDCAN Error Code

HAL_FDCAN_ERROR_NONE
: No error

HAL_FDCAN_ERROR_TIMEOUT
: Timeout error

HAL_FDCAN_ERROR_NOT_INITIALIZED
: Peripheral not initialized

HAL_FDCAN_ERROR_NOT_READY
: Peripheral not ready

HAL_FDCAN_ERROR_NOT_STARTED
: Peripheral not started

HAL_FDCAN_ERROR_NOT_SUPPORTED
: Mode not supported

HAL_FDCAN_ERROR_PARAM
: Parameter error

HAL_FDCAN_ERROR_PENDING
: Pending operation

HAL_FDCAN_ERROR_RAM_ACCESS
: Message RAM Access Failure

HAL_FDCAN_ERROR_FIFO_EMPTY
: Put element in full FIFO

HAL_FDCAN_ERROR_FIFO_FULL
: Get element from empty FIFO

HAL_FDCAN_ERROR_LOG_OVERFLOW
: Overflow of CAN Error Logging Counter

HAL_FDCAN_ERROR_RAM_WDG
: Message RAM Watchdog event occurred

HAL_FDCAN_ERROR_PROTOCOL_ARBT
: Protocol Error in Arbitration Phase (Nominal Bit Time is used)

HAL_FDCAN_ERROR_PROTOCOL_DATA
: Protocol Error in Data Phase (Data Bit Time is used)

HAL_FDCAN_ERROR_RESERVED_AREA
: Access to Reserved Address

HAL state definition

HAL_SMBUS_STATE_RESET

SMBUS not yet initialized or disabled

HAL_SMBUS_STATE_READY

SMBUS initialized and ready for use

HAL_SMBUS_STATE_BUSY

SMBUS internal process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_TX

Master Data Transmission process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_RX

Master Data Reception process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_TX

Slave Data Transmission process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_RX

Slave Data Reception process is ongoing

HAL_SMBUS_STATE_TIMEOUT

Timeout state

HAL_SMBUS_STATE_ERROR

Reception process is ongoing

HAL_SMBUS_STATE_LISTEN

Address Listen Mode is ongoing

Tick Frequency

HAL_TICK_FREQ_10HZ

HAL_TICK_FREQ_100HZ

HAL_TICK_FREQ_1KHZ

HAL_TICK_FREQ_DEFAULT

Boot Mode

SYSCFG_BOOT_MAINFLASH

SYSCFG_BOOT_SYSTEMFLASH

SYSCFG_BOOT_FMC

SYSCFG_BOOT_SRAM

SYSCFG_BOOT_QUADSPI

CCM Write protection

SYSCFG_CCMSRAMWRP_PAGE0

CCMSRAM Write protection page 0

SYSCFG_CCMSRAMWRP_PAGE1

CCMSRAM Write protection page 1

SYSCFG_CCMSRAMWRP_PAGE2

CCMSRAM Write protection page 2

SYSCFG_CCMSRAMWRP_PAGE3

CCMSRAM Write protection page 3

SYSCFG_CCMSRAMWRP_PAGE4

CCMSRAM Write protection page 4

SYSCFG_CCMSRAMWRP_PAGE5

CCMSRAM Write protection page 5

SYSCFG_CCMSRAMWRP_PAGE6

CCMSRAM Write protection page 6

SYSCFG_CCMSRAMWRP_PAGE7

CCMSRAM Write protection page 7

SYSCFG_CCMSRAMWRP_PAGE8

CCMSRAM Write protection page 8

SYSCFG_CCMSRAMWRP_PAGE9

CCMSRAM Write protection page 9

SYSCFG_CCMSRAMWRP_PAGE10

CCMSRAM Write protection page 10

SYSCFG_CCMSRAMWRP_PAGE11

CCMSRAM Write protection page 11

SYSCFG_CCMSRAMWRP_PAGE12

CCMSRAM Write protection page 12

SYSCFG_CCMSRAMWRP_PAGE13

CCMSRAM Write protection page 13

SYSCFG_CCMSRAMWRP_PAGE14

CCMSRAM Write protection page 14

SYSCFG_CCMSRAMWRP_PAGE15

CCMSRAM Write protection page 15

SYSCFG_CCMSRAMWRP_PAGE16

CCMSRAM Write protection page 16

SYSCFG_CCMSRAMWRP_PAGE17

CCMSRAM Write protection page 17

SYSCFG_CCMSRAMWRP_PAGE18

CCMSRAM Write protection page 18

SYSCFG_CCMSRAMWRP_PAGE19

CCMSRAM Write protection page 19

SYSCFG_CCMSRAMWRP_PAGE20

CCMSRAM Write protection page 20

SYSCFG_CCMSRAMWRP_PAGE21

CCMSRAM Write protection page 21

SYSCFG_CCMSRAMWRP_PAGE22

CCMSRAM Write protection page 22

SYSCFG_CCMSRAMWRP_PAGE23

CCMSRAM Write protection page 23

SYSCFG_CCMSRAMWRP_PAGE24

CCMSRAM Write protection page 24

SYSCFG_CCMSRAMWRP_PAGE25

CCMSRAM Write protection page 25

SYSCFG_CCMSRAMWRP_PAGE26

CCMSRAM Write protection page 26

SYSCFG_CCMSRAMWRP_PAGE27

CCMSRAM Write protection page 27

SYSCFG_CCMSRAMWRP_PAGE28

CCMSRAM Write protection page 28

SYSCFG_CCMSRAMWRP_PAGE29

CCMSRAM Write protection page 29

SYSCFG_CCMSRAMWRP_PAGE30

CCMSRAM Write protection page 30

SYSCFG_CCMSRAMWRP_PAGE31

CCMSRAM Write protection page 31

SYSCFG Exported Macros

__HAL_SYSCFG_REMAPMEMORY_FLASH

__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH

__HAL_SYSCFG_REMAPMEMORY_SRAM

__HAL_SYSCFG_REMAPMEMORY_FMC

__HAL_SYSCFG_REMAPMEMORY_QUADSPI

__HAL_SYSCFG_GET_BOOT_MODE

Description:

- Return the boot mode as configured by user.

Return value:

- The: boot mode as configured by user. The returned value can be one of the following values:
  - SYSCFG_BOOT_MAINFLASH
  - SYSCFG_BOOT_SYSTEMFLASH
  - SYSCFG_BOOT_FMC (*)
  - SYSCFG_BOOT_QUADSPI (*)
  - SYSCFG_BOOT_SRAM

Notes:

- (*) availability depends on devices

__HAL_SYSCFG_CCMSRAM_WRP_1_31_ENABLE

Description:

- CCMSRAM page write protection enable macro.

Parameters:

- __CCMSRAMWRP__: This parameter can be a value of

Return value:

- None

Notes:

- write protection can only be disabled by a system reset

__HAL_SYSCFG_CCMSRAM_WRP_0_31_ENABLE

__HAL_SYSCFG_CCMSRAM_WRP_UNLOCK

Notes:

- Writing a wrong key reactivates the write protection

__HAL_SYSCFG_CCMSRAM_ERASE

Notes:

- __SYSCFG_GET_FLAG(SYSCFG_FLAG_CCMSRAM_BUSY) may be used to check end of erase

__HAL_SYSCFG_FPU_INTERRUPT_ENABLE

Description:

- Floating Point Unit interrupt enable/disable macros.

Parameters:

- __INTERRUPT__: This parameter can be a value of

__HAL_SYSCFG_FPU_INTERRUPT_DISABLE

__HAL_SYSCFG_BREAK_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset.

Enable and lock the connection of Flash ECC error connection to TIM1/8/15/16/17 Break input.

__HAL_SYSCFG_BREAK_LOCKUP_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset.

Enable and lock the connection of Cortex-M4 LOCKUP (Hardfault) output to TIM1/8/15/16/17 Break input.

__HAL_SYSCFG_BREAK_PVD_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset.

Enable and lock the PVD connection to Timer1/8/15/16/17 Break input, as well as the PVDE and PLS[2:0] in the PWR_CR2 register.

__HAL_SYSCFG_BREAK_SRAMPARITY_LOCK

Notes:

- The selected configuration is locked and can be unlocked by system reset.

Enable and lock the SRAM parity error (first 32kB of SRAM1 + CCM SRAM) signal connection to TIM1/8/15/16/17 Break input.

__HAL_SYSCFG_GET_FLAG

Description:

- Check SYSCFG flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SYSCFG_FLAG_SRAM_PE SRAM Parity Error Flag
  - SYSCFG_FLAG_CCMSRAM_BUSY CCMSRAM Erase Ongoing

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SYSCFG_CLEAR_FLAG

__HAL_SYSCFG_FASTMODEPLUS_ENABLE

Description:

- Fast-mode Plus driving capability enable/disable macros.

Parameters:

- __FASTMODEPLUS__: This parameter can be a value of :
  - SYSCFG_FASTMODEPLUS_PB6 Fast-mode Plus driving capability activation on PB6
  - SYSCFG_FASTMODEPLUS_PB7 Fast-mode Plus driving capability activation on PB7
  - SYSCFG_FASTMODEPLUS_PB8 Fast-mode Plus driving capability activation on PB8
  - SYSCFG_FASTMODEPLUS_PB9 Fast-mode Plus driving capability activation on PB9

__HAL_SYSCFG_FASTMODEPLUS_DISABLE

Fast-mode Plus on GPIO

SYSCFG_FASTMODEPLUS_PB6

Enable Fast-mode Plus on PB6

SYSCFG_FASTMODEPLUS_PB7

Enable Fast-mode Plus on PB7

SYSCFG_FASTMODEPLUS_PB8

Enable Fast-mode Plus on PB8

SYSCFG_FASTMODEPLUS_PB9

Enable Fast-mode Plus on PB9

Flags

SYSCFG_FLAG_SRAM_PE

SRAM parity error (first 32kB of SRAM1 + CCM SRAM)

SYSCFG_FLAG_CCMSRAM_BUSY

CCMSRAM busy by erase operation

FPU Interrupts

SYSCFG_IT_FPU_IOC

Floating Point Unit Invalid operation Interrupt

SYSCFG_IT_FPU_DZC

Floating Point Unit Divide-by-zero Interrupt

SYSCFG_IT_FPU_UFC

Floating Point Unit Underflow Interrupt

SYSCFG_IT_FPU_OFC

Floating Point Unit Overflow Interrupt

SYSCFG_IT_FPU_IDC

Floating Point Unit Input denormal Interrupt

SYSCFG_IT_FPU_IXC

Floating Point Unit Inexact Interrupt

VREFBUF High Impedance

SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE

VREF_plus pin is internally connected to Voltage reference buffer output

SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE

VREF_plus pin is high impedance

VREFBUF Voltage Scale

SYSCFG_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREFBUF_OUT = 2.048V)

SYSCFG_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREFBUF_OUT = 2.5V)

SYSCFG_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 2 (VREFBUF_OUT = 2.9V)

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_OversamplingTypeDef

ADC_OversamplingTypeDef is defined in the stm32g4xx_hal_adc.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***
- ***uint32_t TriggeredMode***
- ***uint32_t OversamplingStopReset***

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_HAL_EC_OVS_RATIO***
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_HAL_EC_OVS_DISCONT_MODE***
- ***uint32_t ADC_OversamplingTypeDef::OversamplingStopReset***
  Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of ***ADC_HAL_EC_OVS_SCOPE_REG***

7.1.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32g4xx_hal_adc.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t Resolution***
- ***uint32_t DataAlign***
- ***uint32_t GainCompensation***
- ***uint32_t ScanConvMode***
- ***uint32_t EOCSelection***
- ***FunctionalState LowPowerAutoWait***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***FunctionalState DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t SamplingMode***
- ***FunctionalState DMAContinuousRequests***
- ***uint32_t Overrun***
- ***FunctionalState OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***

  Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from system clock or PLL (Refer to reference manual for list of clocks available)) and clock prescaler. This parameter can be a value of ***ADC_HAL_EC_COMMON_CLOCK_SOURCE***. Note: The ADC clock configuration is common to all ADC instances. Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for details. Note: In case of usage of asynchronous clock, the selected clock must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if all ADC instances are disabled.

- ***uint32_t ADC_InitTypeDef::Resolution***

  Configure the ADC resolution. This parameter can be a value of ***ADC_HAL_EC_RESOLUTION***

- ***uint32_t ADC_InitTypeDef::DataAlign***

  Specify ADC data alignment in conversion data register (right or left). Refer to reference manual for alignments formats versus resolutions. This parameter can be a value of ***ADC_HAL_EC_DATA_ALIGN***

- ***uint32_t ADC_InitTypeDef::GainCompensation***

  Specify the ADC gain compensation coefficient to be applied to ADC raw conversion data, based on following formula: DATA = DATA(raw) * (gain compensation coef) / 4096 2.12 bit format, unsigned: 2 bits exponents / 12 bits mantissa Gain step is 1/4096 = 0.000244 Gain range is 0.0000 to 3.999756 This parameter value can be 0 Gain compensation will be disabled and coefficient set to 0 1 -> 0x3FFF Gain compensation will be enabled and coefficient set to specified valueNote: Gain compensation when enabled is appied to all channels.

- ***uint32_t ADC_InitTypeDef::ScanConvMode***

  Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode***

- ***uint32_t ADC_InitTypeDef::EOCSelection***

  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.

- ***FunctionalState ADC_InitTypeDef::LowPowerAutoWait***

  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEx_InjectedGetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since they clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when ADC conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start. (in case of usage of ADC group injected, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).

- ***FunctionalState ADC_InitTypeDef::ContinuousConvMode***

  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfConversion***

  Specify the number of ranks that will be converted within the regular group sequencer. To use the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_InitTypeDef::DiscontinuousConvMode***

  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***

  Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.

- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***

  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.

- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***

  Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***

- ***uint32_t ADC_InitTypeDef::SamplingMode***

  Select the sampling mode to be used for ADC group regular conversion. This parameter can be a value of ***ADC_regular_sampling_mode***

- ***FunctionalState ADC_InitTypeDef::DMAContinuousRequests***

  Specify whether the DMA requests are performed in one shot mode (DMA transfer stops when number of conversions is reached) or in continuous mode (DMA transfer unlimited, whatever number of conversions). This parameter can be set to ENABLE or DISABLE. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached.

- ***uint32_t ADC_InitTypeDef::Overrun***

  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of ***ADC_HAL_EC_REG_OVR_DATA_BEHAVIOR***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:

  – Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.

  – Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).

- ***FunctionalState ADC_InitTypeDef::OversamplingMode***

  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected

- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling***

  Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

7.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32g4xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***
- ***uint32_t SingleDiff***
- ***uint32_t OffsetNumber***
- ***uint32_t Offset***
- ***uint32_t OffsetSign***
- ***FunctionalState OffsetSaturation***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***

  Specify the channel to configure into ADC regular group. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.

- ***uint32_t ADC_ChannelConfTypeDef::Rank***

  Specify the rank in the regular group sequencer. This parameter can be a value of ***ADC_HAL_EC_REG_SEQ_RANKS*** Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***

  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Caution: This parameter applies to a channel that can be used into regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.

- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***

  Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING*** Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)

- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***

  Select the offset number This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB*** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.

- ***uint32_t ADC_ChannelConfTypeDef::Offset***

  Define the offset to be applied on the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***uint32_t ADC_ChannelConfTypeDef::OffsetSign***

  Define if the offset should be substracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSaturation***

  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

7.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32g4xx_hal_adc.h

Data Fields

- ***uint32_t WatchdogNumber***
- ***uint32_t WatchdogMode***
- ***uint32_t Channel***
- ***FunctionalState ITMode***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***
- ***uint32_t FilteringConfig***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***

  Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADC_HAL_EC_AWD_NUMBER***.

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***

  Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel or all channels, ADC groups regular and-or injected. For Analog Watchdog 2 and 3: Several channels can be monitored by applying successively the AWD init structure. Channels on ADC group regular and injected are not differentiated: Set value 'ADC_ANALOGWATCHDOG_SINGLE_xxx' to monitor 1 channel, value 'ADC_ANALOGWATCHDOG_ALL_xxx' to monitor all channels, 'ADC_ANALOGWATCHDOG_NONE' to monitor no channel. This parameter can be a value of ***ADC_analog_watchdog_mode***.

- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***

  Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of ***ADC_HAL_EC_CHANNEL***.

- ***FunctionalState ADC_AnalogWDGConfTypeDef::ITMode***

  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE

- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***

  Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).

- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***

  Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).

- ***uint32_t ADC_AnalogWDGConfTypeDef::FilteringConfig***

  Specify whether filtering should be use and the number of samples to consider. Before setting flag or raising interrupt, analog watchdog can wait to have several consecutive out-of-window samples. This parameter allows to configure this number. This parameter only applies to Analog watchdog 1. For others, use value ADC_AWD_FILTERING_NONE. This parameter can be a value of ***ADC_analog_watchdog_filtering_config***.

7.1.5 ADC_InjectionConfigTypeDef

ADC_InjectionConfigTypeDef is defined in the stm32g4xx_hal_adc.h

Data Fields

- ***uint32_t ContextQueue***
- ***uint32_t ChannelCount***

Field Documentation

- ***uint32_t ADC_InjectionConfigTypeDef::ContextQueue***

  Injected channel configuration context: build-up over each **HAL_ADCEx_InjectedConfigChannel()** call to finally initialize JSQR register at **HAL_ADCEx_InjectedConfigChannel()** last call

- ***uint32_t ADC_InjectionConfigTypeDef::ChannelCount***

  Number of channels in the injected sequence

7.1.6 ADC_HandleTypeDef

ADC_HandleTypeDef is defined in the stm32g4xx_hal_adc.h

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***ADC_InjectionConfigTypeDef InjectionConfig***

Field Documentation

- ***ADC_TypeDef* ADC_HandleTypeDef::Instance***

  Register base address

- ***ADC_InitTypeDef ADC_HandleTypeDef::Init***

  ADC initialization parameters and regular conversions setting

- ***DMA_HandleTypeDef* ADC_HandleTypeDef::DMA_Handle***

  Pointer DMA Handler

- ***HAL_LockTypeDef ADC_HandleTypeDef::Lock***

  ADC locking object

- ***__IO uint32_t ADC_HandleTypeDef::State***

  ADC communication state (bitmap of ADC states)

- ***__IO uint32_t ADC_HandleTypeDef::ErrorCode***

  ADC Error code

- ***ADC_InjectionConfigTypeDef ADC_HandleTypeDef::InjectionConfig***

  ADC injected channel configuration build-up structure

7.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

7.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.
- ADC offset shared on 4 offset instances.
- ADC gain compensation
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

7.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from system clock or PLL (output divider P) running up to 75MHz.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL enable: (optional: if asynchronous clock selected)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLL;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
   - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().

4. Optionally, in case of usage of DMA:
   – Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   – Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
   – Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().
2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   – ADC conversion by polling:
      ◦ Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
      ◦ Wait for ADC conversion completion using function HAL_ADC_PollForConversion()
      ◦ Retrieve conversion results using function HAL_ADC_GetValue()
      ◦ Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   – ADC conversion by interruption:
      ◦ Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
      ◦ Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
      ◦ Retrieve conversion results using function HAL_ADC_GetValue()
      ◦ Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   – ADC conversion with transfer by DMA:
      ◦ Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
      ◦ Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
      ◦ Conversion results are automatically transferred by DMA into destination variable address.
      ◦ Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

Note: Callback functions must be implemented in user program:

- *HAL_ADC_ErrorCallback()*
- *HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)*
- *HAL_ADC_ConvCpltCallback()*
- *HAL_ADC_ConvHalfCpltCallback*

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
     - RCC_OscInitStructure.HSI14State = RCC_HSI14_OFF; (if not used for system clock)
     - HAL_RCC_OscConfig(&RCC_OscInitStructure);
2. ADC pins configuration
   - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   - Deinitialize the DMA using function HAL_DMA_Init().
   - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

Callback registration

The compilation flag USE_HAL_ADC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_ADC_RegisterCallback() to register an interrupt callback.

Function @ref HAL_ADC_RegisterCallback() allows to register following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_ADC_UnRegisterCallback to reset a callback to the default weak function.

@ref HAL_ADC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback

- MspDeInitCallback : ADC Msp DeInit callback

By default, after the @ref HAL_ADC_Init() and when the state is @ref HAL_ADC_STATE_RESET all callbacks are set to the corresponding weak functions: examples @ref HAL_ADC_ConvCpltCallback(), @ref HAL_ADC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_ADC_Init()/ @ref HAL_ADC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the @ref HAL_ADC_Init()/ @ref HAL_ADC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_ADC_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in @ref HAL_ADC_STATE_READY or @ref HAL_ADC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_ADC_RegisterCallback() before calling @ref HAL_ADC_DeInit() or @ref HAL_ADC_Init() function.

When the compilation flag USE_HAL_ADC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

7.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- ***HAL_ADC_ConfigChannel***
- ***HAL_ADC_AnalogWDGConfig***

7.2.4 Peripheral state and errors functions

This subsection provides functions to get in run-time the status of the peripheral.

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- ***HAL_ADC_GetState***
- ***HAL_ADC_GetError***

7.2.5 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- As prerequisite, ADC clock must be configured at RCC top level (refer to description of RCC configuration for ADC in header of this file).
- Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".
- This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".
- Parameters related to common ADC registers (ADC clock mode) are set only if all ADCs are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs.

HAL_ADC_DeInit

Function name

HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)

Function description

Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. (function "HAL_ADC_MspDeInit()" is also called under the same conditions: all ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances). If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common group is still running.
- By default, HAL_ADC_DeInit() set ADC in mode deep power-down: this saves more power by reducing leakage currents and is particularly interesting before entering MCU low-power modes.

HAL_ADC_MspInit

Function name

void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_MspDeInit

Function name

void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)

Function description

DeInitialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- All ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances).

HAL_ADC_Start

Function name

HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled (when multimode feature is available): if ADC is Slave, ADC is enabled but conversion is not started, if ADC is master, ADC is enabled and multimode conversion is started.

HAL_ADC_Stop

Function name

HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for regular group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue().
- This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

HAL_ADC_PollForEvent

Function name

HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description

Poll for ADC event.

Parameters

- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values:
  - ADC_EOSMP_EVENT ADC End of Sampling event
  - ADC_AWD1_EVENT ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 devices)
  - ADC_AWD2_EVENT ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 families)
  - ADC_AWD3_EVENT ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 families)
  - ADC_OVR_EVENT ADC Overrun event
  - ADC_JQOVF_EVENT ADC Injected context queue overflow event
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, data register may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group with interruption.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function according to initialization setting : EOC (end of conversion), EOS (end of sequence), OVR overrun. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_IT() must be called for ADC Slave first, then for ADC Master. For ADC Slave, ADC is enabled only (conversion is not started). For ADC Master, ADC is enabled and multimode conversion is started.
- To guarantee a proper reset of all interruptions once all the needed conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure a correct stop of the IT-based conversions.
- By default, HAL_ADC_Start_IT() does not enable the End Of Sampling interruption. If required (e.g. in case of oversampling with trigger mode), the user must: 1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSMP) 2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP) before calling HAL_ADC_Start_IT().

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADC_Start_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start conversion of regular group and transfer result through DMA.

Parameters

- **hadc:** ADC handle
- **pData:** Destination Buffer address.
- **Length:** Number of data to be transferred from ADC peripheral to memory

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA() is designed for single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStart_DMA() function must be used.

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on ADC group injected. If ADC group injected is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Stop_DMA() function is dedicated to single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStop_DMA() API must be used.

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)

Function description

Get ADC regular group conversion result.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** group regular conversion data

Notes

- Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion).
- This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_StartSampling

Function name

HAL_StatusTypeDef HAL_ADC_StartSampling (ADC_HandleTypeDef * hadc)

Function description

Start ADC conversion sampling phase of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : This function should only be called to start sampling when ADC_SAMPLING_MODE_TRIGGER_CONTROLED sampling mode has been selectedADC_SOFTWARE_START has been selected as trigger source

HAL_ADC_StopSampling

Function name

HAL_StatusTypeDef HAL_ADC_StopSampling (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion sampling phase of regular group and start conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : This function should only be called to stop sampling when ADC_SAMPLING_MODE_TRIGGER_CONTROLED sampling mode has been selectedADC_SOFTWARE_START has been selected as trigger sourceafter sampling has been started using HAL_ADC_StartSampling.

HAL_ADC_IRQHandler

Function name

void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)

Function description

Handle ADC interrupt request.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvCpltCallback

Function name

void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvHalfCpltCallback

Function name

void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion DMA half-transfer callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name

void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 1 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ErrorCallback

Function name

void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)

Function description

ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle parameter "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()".If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)

Function description

Configure a channel to be assigned to ADC group regular.

Parameters

- **hadc:** ADC handle
- **sConfig:** Structure of ADC channel assigned to ADC group regular.

Return values

- **HAL:** status

Notes

- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name

HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)

Function description

Configure the analog watchdog.

Parameters

- **hadc:** ADC handle
- **AnalogWDGConfig:** Structure of ADC analog watchdog configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".
- On this STM32 serie, analog watchdog thresholds can be modified while ADC conversion is on going. In this case, some constraints must be taken into account: the programmed threshold values are effective from the next ADC EOC (end of unitary conversion). Considering that registers write delay may happen due to bus activity, this might cause an uncertainty on the effective timing of the new programmed threshold values.

HAL_ADC_GetState

Function name

uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)

Function description

Return the ADC handle state.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** handle state (bitfield on 32 bits)

Notes

- ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) "

HAL_ADC_GetError

Function name

uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)

Function description

Return the ADC error code.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** error code (bitfield on 32 bits)

ADC_ConversionStop

Function name

HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)

Function description

Stop ADC conversion.

Parameters

- **hadc:** ADC handle
- **ConversionGroup:** ADC group regular and/or injected. This parameter can be one of the following values:
  - ADC_REGULAR_GROUP ADC regular conversion type.
  - ADC_INJECTED_GROUP ADC injected conversion type.
  - ADC_REGULAR_INJECTED_GROUP ADC regular and injected conversion type.

Return values

- **HAL:** status.

ADC_Enable

Function name

HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)

Function description

Enable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_Disable

Function name

HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)

Function description

Disable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC conversions must be stopped.

ADC_DMAConvCplt

Function name

void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAHalfConvCplt

Function name

void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAError

Function name

void ADC_DMAError (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

7.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

7.3.1 ADC

ADC

ADC Analog Watchdog filtering configuration

ADC_AWD_FILTERING_NONE

ADC analog wathdog no filtering, one out-of-window sample is needed to raise flag or interrupt

ADC_AWD_FILTERING_2SAMPLES

ADC analog wathdog 2 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_3SAMPLES

ADC analog wathdog 3 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_4SAMPLES

ADC analog wathdog 4 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_5SAMPLES

ADC analog wathdog 5 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_6SAMPLES

ADC analog wathdog 6 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_7SAMPLES

ADC analog wathdog 7 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_8SAMPLES

ADC analog wathdog 8 consecutives out-of-window samples are needed to raise flag or interrupt

ADC Analog Watchdog Mode

ADC_ANALOGWATCHDOG_NONE

No analog watchdog selected

ADC_ANALOGWATCHDOG_SINGLE_REG

Analog watchdog applied to a regular group single channel

ADC_ANALOGWATCHDOG_SINGLE_INJEC

Analog watchdog applied to an injected group single channel

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

Analog watchdog applied to a regular and injected groups single channel

ADC_ANALOGWATCHDOG_ALL_REG

Analog watchdog applied to regular group all channels

ADC_ANALOGWATCHDOG_ALL_INJEC

Analog watchdog applied to injected group all channels

ADC_ANALOGWATCHDOG_ALL_REGINJEC

Analog watchdog applied to regular and injected groups all channels

ADCx CFGR fields

ADC_CFGR_FIELDS

ADCx CFGR sub fields

ADC_CFGR_FIELDS_2

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV

End of unitary conversion flag

ADC_EOC_SEQ_CONV

End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

ADC peripheral internal error (problem of clocking, enable/disable, erroneous state, ...)

HAL_ADC_ERROR_OVR

Overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_JQOVF

Injected context queue overflow error

ADC Event type

ADC_EOSMP_EVENT

ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 series)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 series)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 series)

ADC_OVR_EVENT

ADC overrun event

ADC_JQOVF_EVENT

ADC Injected Context Queue Overflow event

ADC Exported Constants

ADC_AWD_EVENT

ADC Analog watchdog 1 event: Naming for compatibility with other STM32 devices having only one analog watchdog

ADC flags definition

ADC_FLAG_RDY

ADC Ready flag

ADC_FLAG_EOSMP

ADC End of Sampling flag

ADC_FLAG_EOC

ADC End of Regular Conversion flag

ADC_FLAG_EOS

ADC End of Regular sequence of Conversions flag

ADC_FLAG_OVR

ADC overrun flag

ADC_FLAG_JEOC

ADC End of Injected Conversion flag

ADC_FLAG_JEOS

ADC End of Injected sequence of Conversions flag

ADC_FLAG_AWD1

ADC Analog watchdog 1 flag (main analog watchdog)

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag (additional analog watchdog)

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag (additional analog watchdog)

ADC_FLAG_JQOVF

ADC Injected Context Queue Overflow flag

Analog watchdog - Analog watchdog number

ADC_ANALOGWATCHDOG_1

ADC analog watchdog number 1

ADC_ANALOGWATCHDOG_2

ADC analog watchdog number 2

ADC_ANALOGWATCHDOG_3

ADC analog watchdog number 3

ADC instance - Channel number

ADC_CHANNEL_0

ADC external channel (channel connected to GPIO pin) ADCx_IN0

ADC_CHANNEL_1

ADC external channel (channel connected to GPIO pin) ADCx_IN1

ADC_CHANNEL_2

ADC external channel (channel connected to GPIO pin) ADCx_IN2

ADC_CHANNEL_3

ADC external channel (channel connected to GPIO pin) ADCx_IN3

ADC_CHANNEL_4

ADC external channel (channel connected to GPIO pin) ADCx_IN4

ADC_CHANNEL_5

ADC external channel (channel connected to GPIO pin) ADCx_IN5

ADC_CHANNEL_6

ADC external channel (channel connected to GPIO pin) ADCx_IN6

ADC_CHANNEL_7

ADC external channel (channel connected to GPIO pin) ADCx_IN7

ADC_CHANNEL_8

ADC external channel (channel connected to GPIO pin) ADCx_IN8

ADC_CHANNEL_9

ADC external channel (channel connected to GPIO pin) ADCx_IN9

ADC_CHANNEL_10

ADC external channel (channel connected to GPIO pin) ADCx_IN10

ADC_CHANNEL_11

ADC external channel (channel connected to GPIO pin) ADCx_IN11

ADC_CHANNEL_12

ADC external channel (channel connected to GPIO pin) ADCx_IN12

ADC_CHANNEL_13

ADC external channel (channel connected to GPIO pin) ADCx_IN13

ADC_CHANNEL_14

ADC external channel (channel connected to GPIO pin) ADCx_IN14

ADC_CHANNEL_15

ADC external channel (channel connected to GPIO pin) ADCx_IN15

ADC_CHANNEL_16

ADC external channel (channel connected to GPIO pin) ADCx_IN16

ADC_CHANNEL_17

ADC external channel (channel connected to GPIO pin) ADCx_IN17

ADC_CHANNEL_18

ADC external channel (channel connected to GPIO pin) ADCx_IN18

ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. On this STM32 serie, ADC channel available on all instances but ADC2.

ADC_CHANNEL_TEMPSENSOR_ADC1

ADC internal channel connected to Temperature sensor. On this STM32 serie, ADC channel available only on ADC1 instance.

ADC_CHANNEL_TEMPSENSOR_ADC5

ADC internal channel connected to Temperature sensor. On this STM32 serie, ADC channel available only on ADC5 instance. Refer to device datasheet for ADC5 availaibility

ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda. On this STM32 serie, ADC channel available on all ADC instances but ADC2 & ADC4. Refer to device datasheet for ADC4 availaibility

ADC_CHANNEL_VOPAMP1

ADC internal channel connected to OPAMP1 output. On this STM32 serie, ADC channel available only on ADC1 instance.

ADC_CHANNEL_VOPAMP2

ADC internal channel connected to OPAMP2 output. On this STM32 serie, ADC channel available only on ADC2 instance.

ADC_CHANNEL_VOPAMP3_ADC2

ADC internal channel connected to OPAMP3 output. On this STM32 serie, ADC channel available only on ADC2 instance.

ADC_CHANNEL_VOPAMP3_ADC3

ADC internal channel connected to OPAMP3 output. On this STM32 serie, ADC channel available only on ADC3 instance. Refer to device datasheet for ADC3 availability

ADC_CHANNEL_VOPAMP4

ADC internal channel connected to OPAMP4 output. On this STM32 serie, ADC channel available only on ADC5 instance. Refer to device datasheet for ADC5 availability

ADC_CHANNEL_VOPAMP5

ADC internal channel connected to OPAMP5 output. On this STM32 serie, ADC channel available only on ADC5 instance. Refer to device datasheet for ADC5 availability

ADC_CHANNEL_VOPAMP6

ADC internal channel connected to OPAMP6 output. On this STM32 serie, ADC channel available only on ADC4 instance. Refer to device datasheet for ADC4 availability

Channel - Sampling time

ADC_SAMPLETIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

ADC_SAMPLETIME_6CYCLES_5

Sampling time 6.5 ADC clock cycles

ADC_SAMPLETIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

ADC_SAMPLETIME_24CYCLES_5

Sampling time 24.5 ADC clock cycles

ADC_SAMPLETIME_47CYCLES_5

Sampling time 47.5 ADC clock cycles

ADC_SAMPLETIME_92CYCLES_5

Sampling time 92.5 ADC clock cycles

ADC_SAMPLETIME_247CYCLES_5

Sampling time 247.5 ADC clock cycles

ADC_SAMPLETIME_640CYCLES_5

Sampling time 640.5 ADC clock cycles

ADC_SAMPLETIME_3CYCLES_5

Sampling time 3.5 ADC clock cycles. If selected, this sampling time replaces all sampling time 2.5 ADC clock cycles. These 2 sampling times cannot be used simultaneously.

Channel - Single or differential ending

ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

ADC common - Clock source

ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock without prescaler

ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock with prescaler division by 2

ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC conversion data alignment

ADC_DATAALIGN_RIGHT

ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)

ADC_DATAALIGN_LEFT

ADC conversion data alignment: left aligned (aligment on data register MSB bit 15)

ADC instance - Groups

ADC_REGULAR_GROUP

ADC group regular (available on all STM32 devices)

ADC_INJECTED_GROUP

ADC group injected (not available on all STM32 devices)

ADC_REGULAR_INJECTED_GROUP

ADC both groups regular and injected

Multimode - DMA transfer mode depending on ADC resolution

ADC_DMAACCESSMODE_DISABLED

DMA multimode disabled: each ADC uses its own DMA channel

ADC_DMAACCESSMODE_12_10_BITS

DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 12 and 10 bits resolution

ADC_DMAACCESSMODE_8_6_BITS

DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 8 and 6 bits resolution

Multimode - Mode

ADC_MODE_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

ADC_DUALMODE_REGSIMULT

ADC dual mode enabled: group regular simultaneous

ADC_DUALMODE_INTERL

ADC dual mode enabled: Combined group regular interleaved

ADC_DUALMODE_INJECSIMULT

ADC dual mode enabled: group injected simultaneous

ADC_DUALMODE_ALTERTRIG

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

ADC_DUALMODE_REGSIMULT_INJECSIMULT

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

ADC_DUALMODE_REGSIMULT_ALTERTRIG

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

ADC_DUALMODE_REGINTERL_INJECSIMULT

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

ADC_TWOSAMPLINGDELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

ADC_TWOSAMPLINGDELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

ADC_TWOSAMPLINGDELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

ADC_TWOSAMPLINGDELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

ADC_TWOSAMPLINGDELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

ADC_TWOSAMPLINGDELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

ADC_TWOSAMPLINGDELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

ADC_TWOSAMPLINGDELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

ADC_TWOSAMPLINGDELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

ADC_TWOSAMPLINGDELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles

ADC_TWOSAMPLINGDELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles

ADC_TWOSAMPLINGDELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles

ADC instance - Offset number

ADC_OFFSET_NONE

ADC offset disabled: no offset correction for the selected ADC channel

ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Oversampling - Discontinuous mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

ADC_TRIGGEREDMODE_MULTI_TRIGGER

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

ADC_OVERSAMPLING_RATIO_2

ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_4

ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_8

ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_16

ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_32

ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_64

ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_128

ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_256

ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

Oversampling - Oversampling scope for ADC group regular

ADC_REGOVERSAMPLING_CONTINUED_MODE

Oversampling buffer maintained during injection sequence

ADC_REGOVERSAMPLING_RESUMED_MODE

Oversampling buffer zeroed during injection sequence

Oversampling - Data shift

ADC_RIGHTBITSHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_1

ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_2

ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_3

ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_4

ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_5

ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_6

ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_7

ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_8

ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

ADC group regular - Overrun behavior on conversion data

ADC_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

ADC_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer ranks

ADC_REGULAR_RANK_1

ADC group regular sequencer rank 1

ADC_REGULAR_RANK_2

ADC group regular sequencer rank 2

ADC_REGULAR_RANK_3

ADC group regular sequencer rank 3

ADC_REGULAR_RANK_4

ADC group regular sequencer rank 4

ADC_REGULAR_RANK_5

ADC group regular sequencer rank 5

ADC_REGULAR_RANK_6

ADC group regular sequencer rank 6

ADC_REGULAR_RANK_7

ADC group regular sequencer rank 7

ADC_REGULAR_RANK_8

ADC group regular sequencer rank 8

ADC_REGULAR_RANK_9

ADC group regular sequencer rank 9

ADC_REGULAR_RANK_10

ADC group regular sequencer rank 10

ADC_REGULAR_RANK_11

ADC group regular sequencer rank 11

ADC_REGULAR_RANK_12

ADC group regular sequencer rank 12

ADC_REGULAR_RANK_13

ADC group regular sequencer rank 13

ADC_REGULAR_RANK_14

ADC group regular sequencer rank 14

ADC_REGULAR_RANK_15

ADC group regular sequencer rank 15

ADC_REGULAR_RANK_16

ADC group regular sequencer rank 16

ADC instance - Resolution

ADC_RESOLUTION_12B

ADC resolution 12 bits

ADC_RESOLUTION_10B

ADC resolution 10 bits

ADC_RESOLUTION_8B

ADC resolution 8 bits

ADC_RESOLUTION_6B

ADC resolution 6 bits

HAL ADC macro to manage HAL ADC handle, IT and flags.

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

- Check whether the specified ADC flag is set or not.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF ADC Injected Context Queue Overflow flag.

Return value:

- State: of flag (TRUE or FALSE).

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the specified ADC flag.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF ADC Injected Context Queue Overflow flag.

Return value:

- None

HAL ADC helper macro

__HAL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1 (8)
  - ADC_CHANNEL_2 (8)
  - ADC_CHANNEL_3 (8)
  - ADC_CHANNEL_4 (8)
  - ADC_CHANNEL_5 (8)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (7)
  - ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - ADC_CHANNEL_VBAT (6)
  - ADC_CHANNEL_VOPAMP1 (1)
  - ADC_CHANNEL_VOPAMP2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - ADC_CHANNEL_VOPAMP4 (5)
  - ADC_CHANNEL_VOPAMP5 (5)
  - ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __HAL_ADC_CHANNEL_TO_DECIMAL_NB(ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__HAL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1 (8)
  - ADC_CHANNEL_2 (8)
  - ADC_CHANNEL_3 (8)
  - ADC_CHANNEL_4 (8)
  - ADC_CHANNEL_5 (8)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (7)
  - ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - ADC_CHANNEL_VBAT (6)
  - ADC_CHANNEL_VOPAMP1 (1)
  - ADC_CHANNEL_VOPAMP2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - ADC_CHANNEL_VOPAMP4 (5)
  - ADC_CHANNEL_VOPAMP5 (5)
  - ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution. (1, 2, 3, 4, 5, 7) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes:

- Example: __HAL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "ADC_CHANNEL_4".

__HAL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1 (8)
  - ADC_CHANNEL_2 (8)
  - ADC_CHANNEL_3 (8)
  - ADC_CHANNEL_4 (8)
  - ADC_CHANNEL_5 (8)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (7)
  - ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - ADC_CHANNEL_VBAT (6)
  - ADC_CHANNEL_VOPAMP1 (1)
  - ADC_CHANNEL_VOPAMP2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - ADC_CHANNEL_VOPAMP4 (5)
  - ADC_CHANNEL_VOPAMP5 (5)
  - ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): ADC_CHANNEL_1, ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1 (8)
  - ADC_CHANNEL_2 (8)
  - ADC_CHANNEL_3 (8)
  - ADC_CHANNEL_4 (8)
  - ADC_CHANNEL_5 (8)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (7)
  - ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - ADC_CHANNEL_VBAT (6)
  - ADC_CHANNEL_VOPAMP1 (1)
  - ADC_CHANNEL_VOPAMP2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - ADC_CHANNEL_VOPAMP4 (5)
  - ADC_CHANNEL_VOPAMP5 (5)
  - ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__HAL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_VREFINT (7)
  - ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - ADC_CHANNEL_VBAT (6)
  - ADC_CHANNEL_VOPAMP1 (1)
  - ADC_CHANNEL_VOPAMP2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - ADC_CHANNEL_VOPAMP4 (5)
  - ADC_CHANNEL_VOPAMP5 (5)
  - ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__HAL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__HAL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__HAL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref + and Vref- (refer to reference manual).

__HAL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__HAL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__HAL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 serie, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 serie, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32G4, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32G4, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

ADC group injected trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

Injected conversions hardware trigger detection disabled

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

Injected conversions hardware trigger detection on the rising edge

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

Injected conversions hardware trigger detection on the falling edge

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

Injected conversions hardware trigger detection on both the rising and falling edges

ADC group injected trigger source

ADC_INJECTED_SOFTWARE_START

Software triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_CC3

ADC group injected conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_CC4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T2_CC1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T4_CC3

ADC group injected conversion trigger from external peripheral: TIM4 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T4_CC4

ADC group injected conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T7_TRGO

ADC group injected conversion trigger from external peripheral: TIM7 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_CC2

ADC group injected conversion trigger from external peripheral: TIM8 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_CC4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T16_CC1

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T20_TRGO

ADC group injected conversion trigger from external peripheral: TIM20 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T20_TRGO2

ADC group injected conversion trigger from external peripheral: TIM20 TRGO2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T20_CC2

ADC group injected conversion trigger from external peripheral: TIM20 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T20_CC4

ADC group injected conversion trigger from external peripheral: TIM20 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG1

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG2

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG3

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 3 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG4

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 4 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG5

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 5 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG6

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 6 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG7

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 7 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG8

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 8 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG9

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 9 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HRTIM_TRG10

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 10 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_EXT_IT3

ADC group injected conversion trigger from external peripheral: external interrupt line 3. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_EXT_IT15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM_OUT

ADC group injected conversion trigger from external peripheral: LPTIMER OUT event. Trigger edge set to rising edge (default setting).

ADC group injected - Sequencer ranks

ADC_INJECTED_RANK_1

ADC group injected sequencer rank 1

ADC_INJECTED_RANK_2

ADC group injected sequencer rank 2

ADC_INJECTED_RANK_3

ADC group injected sequencer rank 3

ADC_INJECTED_RANK_4

ADC group injected sequencer rank 4

ADC interrupts definition

ADC_IT_RDY

ADC Ready interrupt source

ADC_IT_EOSMP

ADC End of sampling interrupt source

ADC_IT_EOC

ADC End of regular conversion interrupt source

ADC_IT_EOS

ADC End of regular sequence of conversions interrupt source

ADC_IT_OVR

ADC overrun interrupt source

ADC_IT_JEOC

ADC End of injected conversion interrupt source

ADC_IT_JEOS

ADC End of injected sequence of conversions interrupt source

ADC_IT_AWD1

ADC Analog watchdog 1 interrupt source (main analog watchdog)

ADC_IT_AWD2

ADC Analog watchdog 2 interrupt source (additional analog watchdog)

ADC_IT_AWD3

ADC Analog watchdog 3 interrupt source (additional analog watchdog)

ADC_IT_JQOVF

ADC Injected Context Queue Overflow interrupt source

ADC_IT_AWD

ADC Analog watchdog 1 interrupt source: naming for compatibility with other STM32 devices having only one analog watchdog

ADC group regular trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGCONVEDGE_NONE

Regular conversions hardware trigger detection disabled

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC group regular conversion trigger polarity set to rising edge

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC group regular conversion trigger polarity set to falling edge

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular trigger source

ADC_SOFTWARE_START

ADC group regular conversion trigger internal: SW start.

ADC_EXTERNALTRIG_T1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_CC1

ADC group regular conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_CC2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_CC3

ADC group regular conversion trigger from external peripheral: TIM2 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_CC1

ADC group regular conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_CC4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_CC1

ADC group regular conversion trigger from external peripheral: TIM4 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_CC4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T7_TRGO

ADC group regular conversion trigger from external peripheral: TIM7 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_CC1

ADC group regular conversion trigger from external peripheral: TIM8 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T20_TRGO

ADC group regular conversion trigger from external peripheral: TIM20 TRGO. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T20_TRGO2

ADC group regular conversion trigger from external peripheral: TIM20 TRGO2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T20_CC1

ADC group regular conversion trigger from external peripheral: TIM20 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T20_CC2

ADC group regular conversion trigger from external peripheral: TIM20 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T20_CC3

ADC group regular conversion trigger from external peripheral: TIM20 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG1

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG2

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG3

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 3 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG4

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 4 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG5

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 5 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG6

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 6 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG7

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 7 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG8

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 8 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG9

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 9 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HRTIM_TRG10

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 10 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_EXT_IT2

ADC group regular conversion trigger from external peripheral: external interrupt line 2. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_EXT_IT11

ADC group regular conversion trigger from external peripheral: external interrupt line 11. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM_OUT

ADC group regular conversion trigger from external peripheral: LPTIMER OUT event. Trigger edge set to rising edge (default setting).

ADC group regular sampling mode

ADC_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

ADC_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

ADC_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: Trigger rising edge = start sampling Trigger falling edge = stop sampling and start conversion

ADC sequencer scan mode

ADC_SCAN_DISABLE

Scan mode disabled

ADC_SCAN_ENABLE

Scan mode enabled

ADCx SMPR1 fields

ADC_SMPR1_FIELDS

ADC States

HAL_ADC_STATE_RESET

Notes:

- ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy due to an internal process (initialization, calibration)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_REG_EOC

Conversion data available on group regular

HAL_ADC_STATE_REG_OVR

Overrun occurrence

HAL_ADC_STATE_REG_EOSMP

Not available on this STM32 serie: End Of Sampling flag raised

HAL_ADC_STATE_INJ_BUSY

A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_INJ_EOC

Conversion data available on group injected

HAL_ADC_STATE_INJ_JQOVF

Injected queue overflow occurrence

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

ADC in multimode slave state, controlled by another ADC master (when feature available)

8 HAL ADC Extension Driver

8.1 ADCEx Firmware driver registers structures

8.1.1 ADC_InjOversamplingTypeDef

ADC_InjOversamplingTypeDef is defined in the stm32g4xx_hal_adc_ex.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***

Field Documentation

- ***uint32_t ADC_InjOversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_HAL_EC_OVS_RATIO***
- ***uint32_t ADC_InjOversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***

8.1.2 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32g4xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedOffsetSign***
- ***FunctionalState InjectedOffsetSaturation***
- ***uint32_t InjectedNbrOfConversion***
- ***FunctionalState InjectedDiscontinuousConvMode***
- ***FunctionalState AutoInjectedConv***
- ***FunctionalState QueueInjectedContext***
- ***uint32_t ExternalTrigInjecConv***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***FunctionalState InjecOversamplingMode***
- ***ADC_InjOversamplingTypeDef InjecOversampling***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Specifies the channel to configure into ADC group injected. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Specifies the rank in the ADC group injected sequencer. This parameter must be a value of ***ADC_INJ_SEQ_RANKS***. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***

  Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***

  Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***

  Selects the offset number. This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB***. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***

  Defines the offset to be applied on the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetSign***

  Define if the offset should be substracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSaturation***

  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***

  Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***

  Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::AutoInjectedConv***

  Enables or disables the selected ADC group injected automatic conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_INJECTED_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::QueueInjectedContext***

  Specifies whether the context queue feature is enabled. This parameter can be set to ENABLE or DISABLE. If context queue is enabled, injected sequencer&channels configurations are queued on up to 2 contexts. If a new injected context is set when queue is full, error is triggered by interruption and through function 'HAL_ADCEx_InjectedQueueOverflowCallback'. Caution: This feature request that the sequence is fully configured before injected conversion start. Therefore, configure channels with as many calls to **HAL_ADCEx_InjectedConfigChannel()** as the 'InjectedNbrOfConversion' parameter. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***

  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_injected_external_trigger_source***. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***

  Selects the external trigger edge of injected group. This parameter can be a value of ***ADC_injected_external_trigger_edge***. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjecOversamplingMode***

  Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).
- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***

  Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

8.1.3 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32g4xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t DMAAccessMode***
- ***uint32_t TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode***

  Configures the ADC to operate in independent or multimode. This parameter can be a value of ***ADC_HAL_EC_MULTI_MODE***.
- ***uint32_t ADC_MultiModeTypeDef::DMAAccessMode***

  Configures the DMA mode for multimode ADC: selection whether 2 DMA channels (each ADC uses its own DMA channel) or 1 DMA channel (one DMA channel for both ADC, DMA of ADC master) This parameter can be a value of ***ADC_HAL_EC_MULTI_DMA_TRANSFER_RESOLUTION***.

- ***uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay***
  Configures the Delay between 2 sampling phases. This parameter can be a value of ***ADC_HAL_EC_MULTI_TWOSMP_DELAY***. Delay range depends on selected resolution: from 1 to 12 clock cycles for 12 bits, from 1 to 10 clock cycles for 10 bits, from 1 to 8 clock cycles for 8 bits, from 1 to 6 clock cycles for 6 bits.

8.2 ADCEx Firmware driver API description

The following section lists the various functions of the ADCEx library.

8.2.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.
- Set calibration factors for single or differential ending.
- Start conversion of ADC group injected.
- Stop conversion of ADC group injected.
- Poll for conversion complete on ADC group injected.
- Get result of ADC group injected channel conversion.
- Start conversion of ADC group injected and enable interruptions.
- Stop conversion of ADC group injected and disable interruptions.
- When multimode feature is available, start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.

This section contains the following APIs:

- ***HAL_ADCEx_Calibration_Start***
- ***HAL_ADCEx_Calibration_GetValue***
- ***HAL_ADCEx_Calibration_SetValue***
- ***HAL_ADCEx_InjectedStart***
- ***HAL_ADCEx_InjectedStop***
- ***HAL_ADCEx_InjectedPollForConversion***
- ***HAL_ADCEx_InjectedStart_IT***
- ***HAL_ADCEx_InjectedStop_IT***
- ***HAL_ADCEx_MultiModeStart_DMA***
- ***HAL_ADCEx_MultiModeStop_DMA***
- ***HAL_ADCEx_MultiModeGetValue***
- ***HAL_ADCEx_InjectedGetValue***
- ***HAL_ADCEx_InjectedConvCpltCallback***
- ***HAL_ADCEx_InjectedQueueOverflowCallback***
- ***HAL_ADCEx_LevelOutOfWindow2Callback***
- ***HAL_ADCEx_LevelOutOfWindow3Callback***
- ***HAL_ADCEx_EndOfSamplingCallback***
- ***HAL_ADCEx_RegularStop***
- ***HAL_ADCEx_RegularStop_IT***
- ***HAL_ADCEx_RegularStop_DMA***
- ***HAL_ADCEx_RegularMultiModeStop_DMA***

8.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode when multimode feature is available

- Enable or Disable Injected Queue
- Disable ADC voltage regulator
- Enter ADC deep-power-down mode

This section contains the following APIs:

- ***HAL_ADCEx_InjectedConfigChannel***
- ***HAL_ADCEx_MultiModeConfigChannel***
- ***HAL_ADCEx_EnableInjectedQueue***
- ***HAL_ADCEx_DisableInjectedQueue***
- ***HAL_ADCEx_DisableVoltageRegulator***
- ***HAL_ADCEx_EnterADCDeepPowerDownMode***

8.2.3 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).

Parameters

- **hadc:** ADC handle
- **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **HAL:** status

HAL_ADCEx_Calibration_GetValue

Function name

uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

Get the calibration factor.

Parameters

- **hadc:** ADC handle.
- **SingleDiff:** This parameter can be only:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **Calibration:** value.

HAL_ADCEx_Calibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set the calibration factor to overwrite automatic conversion result.

Parameters

- **hadc:** ADC handle
- **SingleDiff:** This parameter can be only:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
- **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)

Return values

- **HAL:** state

HAL_ADCEx_InjectedStart

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled when multimode feature is available: HAL_ADCEx_InjectedStart() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- In case of multimode enabled (when multimode feature is available), HAL_ADCEx_InjectedStop() must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

HAL_ADCEx_InjectedPollForConversion

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for injected group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- Depending on hadc->Init.EOCSelection, JEOS or JEOC is checked and cleared depending on AUTDLY bit status.

HAL_ADCEx_InjectedStart_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group with interruption.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function according to initialization setting : JEOC (end of conversion) or JEOS (end of sequence)
- Case of multimode enabled (when multimode feature is enabled): HAL_ADCEx_InjectedStart_IT() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels, disable interruption of end-of-conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- Case of multimode enabled (when multimode feature is available): HAL_ADCEx_InjectedStop_IT() API must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).
- In case of auto-injection mode, HAL_ADC_Stop() must be used.

HAL_ADCEx_MultiModeStart_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start MultiMode conversion and transfer regular results through DMA.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes).

Return values

- **HAL:** status

Notes

- Multimode must have been previously configured using HAL_ADCEx_MultiModeConfigChannel() function. Interruptions enabled in this function: overrun, DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- State field of Slave ADC handle is not updated in this configuration: user should not rely on it for information related to Slave regular conversions.

HAL_ADCEx_MultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. MultiMode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable Multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_MultiModeGetValue

Function name

uint32_t HAL_ADCEx_MultiModeGetValue (ADC_HandleTypeDef * hadc)

Function description

Return the last ADC Master and Slave regular conversions results when in multimode configuration.

Parameters

- **hadc:** ADC handle of ADC Master (handle of ADC Slave must not be used)

Return values

- **The:** converted data values.

HAL_ADCEx_InjectedGetValue

Function name

uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)

Function description

Get ADC injected group conversion result.

Parameters

- **hadc:** ADC handle
- **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values:
  - ADC_INJECTED_RANK_1 ADC group injected rank 1
  - ADC_INJECTED_RANK_2 ADC group injected rank 2
  - ADC_INJECTED_RANK_3 ADC group injected rank 3
  - ADC_INJECTED_RANK_4 ADC group injected rank 4

Return values

- **ADC:** group injected conversion data

Notes

- Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion).
- This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC.If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32 families). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_InjectedConvCpltCallback

Function name

void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Injected conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_InjectedQueueOverflowCallback

Function name

void HAL_ADCEx_InjectedQueueOverflowCallback (ADC_HandleTypeDef * hadc)

Function description

Injected context queue overflow callback.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- This callback is called if injected context queue is enabled (parameter "QueueInjectedContext" in injected channel configuration) and if a new injected context is set when queue is full (maximum 2 contexts).

HAL_ADCEx_LevelOutOfWindow2Callback

Function name

void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 2 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_LevelOutOfWindow3Callback

Function name

void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 3 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_EndOfSamplingCallback

Function name

void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)

Function description

End Of Sampling callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_RegularStop

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of ADC groups regular and injected, disable interrution of end-of-conversion, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- HAL_ADCEx_RegularStop_DMA() function is dedicated to single-ADC mode only. For multimode (when multimode feature is available), HAL_ADCEx_RegularMultiModeStop_DMA() API must be used.

HAL_ADCEx_RegularMultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularMultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop DMA-based multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral if no injected conversion is on-going.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. Multimode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADCEx_RegularStop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_InjectedConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)

Function description

Configure a channel to be assigned to ADC group injected.

Parameters

- **hadc:** ADC handle
- **sConfigInjected:** Structure of ADC injected group and ADC channel for injected group.

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_InjectionConfTypeDef".
- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Caution: For Injected Context Queue use, a context must be fully defined before start of injected conversion. All channels are configured consecutively for the same ADC instance. Therefore, the number of calls to HAL_ADCEx_InjectedConfigChannel() must be equal to the value of parameter InjectedNbrOfConversion for each context. Example 1: If 1 context is intended to be used (or if there is no use of the Injected Queue Context feature) and if the context contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel (i.e. 3 times) before starting a conversion. This function must not be called to configure a 4th injected channel: it would start a new context into context queue.Example 2: If 2 contexts are intended to be used and each of them contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel and for each context (3 channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set, that is after the first three HAL_ADCEx_InjectedConfigChannel() calls. The 2nd context can be set on the fly.

HAL_ADCEx_MultiModeConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * multimode)

Function description

Enable ADC multimode and configure multimode parameters.

Parameters

- **hadc:** Master ADC handle
- **multimode:** Structure of ADC multimode configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without resetting the ADCs. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".
- To move back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

HAL_ADCEx_EnableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_EnableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Enable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This function resets CFGR register JQDIS bit in order to enable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Disable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This function sets CFGR register JQDIS bit in order to disable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableVoltageRegulator

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableVoltageRegulator (ADC_HandleTypeDef * hadc)

Function description

Disable ADC voltage regulator.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Disabling voltage regulator allows to save power. This operation can be carried out only when ADC is disabled.
- To enable again the voltage regulator, the user is expected to resort to HAL_ADC_Init() API.

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name

HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)

Function description

Enter ADC deep-power-down mode.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This mode is achieved in setting DEEPPWD bit and allows to save power in reducing leakage currents. It is particularly interesting before entering stop modes.
- Setting DEEPPWD automatically clears ADVREGEN bit and disables the ADC voltage regulator. This means that this API encompasses HAL_ADCEx_DisableVoltageRegulator(). Additionally, the internal calibration is lost.
- To exit the ADC deep-power-down mode, the user is expected to resort to HAL_ADC_Init() API as well as to relaunch a calibration with HAL_ADCEx_Calibration_Start() API or to re-apply a previously saved calibration factor.

8.3 ADCEx Firmware driver defines

The following section lists the various define and macros of the module.

8.3.1 ADCEx

ADCEx

ADC Extended Exported Macros

ADC_FORCE_MODE_INDEPENDENT

Description:

- Force ADC instance in multimode mode independent (multimode disable).

Parameters:

- __HANDLE__: ADC handle.

Return value:

- None

Notes:

- This macro must be used only in case of transition from multimode to mode independent and in case of unknown previous state, to ensure ADC configuration is in mode independent. Standard way of multimode configuration change is done from HAL ADC handle of ADC master using function "HAL_ADCEx_MultiModeConfigChannel(..., ADC_MODE_INDEPENDENT)")". Usage of this macro is not the Standard way of multimode configuration and can lead to have HAL ADC handles status misaligned. Usage of this macro must be limited to cases mentionned above.

ADC Extended Offset Sign

ADC_OFFSET_SIGN_NEGATIVE

Offset sign negative, offset is substracted

ADC_OFFSET_SIGN_POSITIVE

Offset sign positive, offset is added

9 HAL COMP Generic Driver

9.1 COMP Firmware driver registers structures

9.1.1 COMP_InitTypeDef

COMP_InitTypeDef is defined in the stm32g4xx_hal_comp.h

Data Fields

- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t Hysteresis***
- ***uint32_t OutputPol***
- ***uint32_t BlankingSrce***
- ***uint32_t TriggerMode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::InputPlus***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_InputPlus***
- ***uint32_t COMP_InitTypeDef::InputMinus***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_InputMinus***
- ***uint32_t COMP_InitTypeDef::Hysteresis***
  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_Hysteresis***
- ***uint32_t COMP_InitTypeDef::OutputPol***
  Set comparator output polarity. This parameter can be a value of ***COMP_OutputPolarity***
- ***uint32_t COMP_InitTypeDef::BlankingSrce***
  Set comparator blanking source. This parameter can be a value of ***COMP_BlankingSrce***
- ***uint32_t COMP_InitTypeDef::TriggerMode***
  Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of ***COMP_EXTI_TriggerMode***

9.1.2 COMP_HandleTypeDef

COMP_HandleTypeDef is defined in the stm32g4xx_hal_comp.h

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_COMP_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***COMP_TypeDef* COMP_HandleTypeDef::Instance***
  Register base address
- ***COMP_InitTypeDef COMP_HandleTypeDef::Init***
  COMP required parameters
- ***HAL_LockTypeDef COMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_COMP_StateTypeDef COMP_HandleTypeDef::State***
  COMP communication state
- ***__IO uint32_t COMP_HandleTypeDef::ErrorCode***
  COMP error code

9.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

9.2.1 COMP Peripheral features

The STM32G4xx device family integrates seven analog comparators instances: COMP1, COMP2, COMP3, COMP4, COMP5, COMP6 and COMP7.

1. Comparators input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. Comparators output level is available using HAL_COMP_GetOutputLevel() and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. The comparators have interrupt capability through the EXTI controller with wake-up from sleep and stop modes. From the corresponding IRQ handler, the right interrupt source can be retrieved using macro __HAL_COMP_COMPx_EXTI_GET_FLAG().

9.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32G4xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the HAL_COMP_MspInit():
   - Configure the GPIO connected to comparator inputs plus and minus in analog mode using HAL_GPIO_Init().
   - If needed, configure the GPIO connected to comparator output in alternate function mode using HAL_GPIO_Init().
   - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL_GPIO_Init() function. After that enable the comparator interrupt vector using HAL_NVIC_EnableIRQ() function.
2. Configure the comparator using HAL_COMP_Init() function:
   - Select the input minus (inverting input)
   - Select the input plus (non-inverting input)
   - Select the hysteresis
   - Select the blanking source
   - Select the output polarity

Note: HAL_COMP_Init() calls internally __HAL_RCC_SYSCFG_CLK_ENABLE() to enable internal control clock of the comparators. However, this is a legacy strategy. In future STM32 families, COMP clock enable must be implemented by user in "HAL_COMP_MspInit()". Therefore, for compatibility anticipation, it is recommended to implement __HAL_RCC_SYSCFG_CLK_ENABLE() in "HAL_COMP_MspInit()".

3. Reconfiguration on-the-fly of comparator can be done by calling again function HAL_COMP_Init() with new input structure parameters values.
4. Enable the comparator using HAL_COMP_Start() function.
5. Use HAL_COMP_TriggerCallback() or HAL_COMP_GetOutputLevel() functions to manage comparator outputs (events and output level).
6. Disable the comparator using HAL_COMP_Stop() function.
7. De-initialize the comparator using HAL_COMP_DeInit() function.
8. For safety purpose, comparator configuration can be locked using HAL_COMP_Lock() function. The only way to unlock the comparator is a device hardware reset.

Callback registration

The compilation flag USE_HAL_COMP_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_COMP_RegisterCallback() to register an interrupt callback.

Function HAL_COMP_RegisterCallback() allows to register following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.

- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_COMP_UnRegisterCallback to reset a callback to the default weak function.

HAL_COMP_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_COMP_Init() and when the state is HAL_COMP_STATE_RESET all callbacks are set to the corresponding weak functions: example HAL_COMP_TriggerCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_COMP_Init()/ HAL_COMP_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_COMP_Init()/ HAL_COMP_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_COMP_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_COMP_STATE_READY or HAL_COMP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_COMP_RegisterCallback() before calling HAL_COMP_DeInit() or HAL_COMP_Init() function.

When the compilation flag USE_HAL_COMP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

9.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- ***HAL_COMP_Init***
- ***HAL_COMP_DeInit***
- ***HAL_COMP_MspInit***
- ***HAL_COMP_MspDeInit***

9.2.4 IO operation functions

This section provides functions allowing to:

- Start a comparator instance.
- Stop a comparator instance.

This section contains the following APIs:

- ***HAL_COMP_Start***
- ***HAL_COMP_Stop***
- ***HAL_COMP_IRQHandler***

9.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the comparators.

This section contains the following APIs:

- ***HAL_COMP_Lock***
- ***HAL_COMP_GetOutputLevel***
- ***HAL_COMP_TriggerCallback***

9.2.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:

- ***HAL_COMP_GetState***
- ***HAL_COMP_GetError***

9.2.7 Detailed description of functions

HAL_COMP_Init

Function name

HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_COMP_DeInit

Function name

HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP peripheral.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name

void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_MspDeInit

Function name

void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_Start

Function name

HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop

Function name

HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)

Function description

Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_IRQHandler

Function name

void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)

Function description

Comparator IRQ handler.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_Lock

Function name

HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)

Function description

Lock the selected comparator configuration.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- A system reset is required to unlock the comparator configuration.
- Locking the comparator from reset state is possible if __HAL_RCC_SYSCFG_CLK_ENABLE() is being called before.

HAL_COMP_GetOutputLevel

Function name

uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)

Function description

Return the output level (high or low) of the selected comparator.

HAL_COMP_TriggerCallback

Function name

void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)

Function description

Comparator trigger callback.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_GetState

Function name

HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)

Function description

Return the COMP handle state.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** state

HAL_COMP_GetError

Function name

uint32_t HAL_COMP_GetError (COMP_HandleTypeDef * hcomp)

Function description

Return the COMP error code.

Parameters

- **hcomp:** COMP handle

Return values

- **COMP:** error code

9.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

9.3.1 COMP

COMP

COMP blanking source

COMP_BLANKINGSRC_NONE

Comparator output without blanking

COMP_BLANKINGSRC_TIM1_OC5_COMP1

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP1). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM1_OC5_COMP2

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP2). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM1_OC5_COMP3

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP3). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM1_OC5_COMP4

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP4). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM1_OC5_COMP5

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP5). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM1_OC5_COMP6

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP6). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM1_OC5_COMP7

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP7). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM2_OC3_COMP1

Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP1). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM2_OC3_COMP2

Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP2). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM2_OC3_COMP5

Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP5). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM2_OC4_COMP3

Comparator output blanking source TIM2 OC4 (specific to COMP instance: COMP3). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM2_OC4_COMP6

Comparator output blanking source TIM2 OC4 (specific to COMP instance: COMP6). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM3_OC3_COMP1

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP1). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM3_OC3_COMP2

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP2). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM3_OC3_COMP3

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP3). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM3_OC3_COMP5

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP5). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM3_OC3_COMP7

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP7). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM3_OC4_COMP4

Comparator output blanking source TIM3 OC4 (specific to COMP instance: COMP4). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP1

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP1). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP2

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP2). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP3

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP3). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP4

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP4). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP5

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP5). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP6

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP6). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM8_OC5_COMP7

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP7). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM15_OC1_COMP4

Comparator output blanking source TIM15 OC1 (specific to COMP instance: COMP4). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM15_OC2_COMP6

Comparator output blanking source TIM15 OC2 (specific to COMP instance: COMP6). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM15_OC2_COMP7

Comparator output blanking source TIM15 OC3 (specific to COMP instance: COMP7). Note: For COMPx & TIMx instances availability, please refer to datasheet

COMP_BLANKINGSRC_TIM20_OC5

Comparator output blanking source TIM20 OC5 (Common to all COMP instances)

COMP_BLANKINGSRC_TIM15_OC1

Comparator output blanking source TIM15 OC1 (Common to all COMP instances)

COMP_BLANKINGSRC_TIM4_OC3

Comparator output blanking source TIM4 OC3 (Common to all COMP instances)

COMP Error Code

HAL_COMP_ERROR_NONE

No error

COMP Exported Types

COMP_STATE_BITFIELD_LOCK

COMP EXTI Lines

COMP_EXTI_LINE_COMP1

EXTI line 21 connected to COMP1 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_LINE_COMP2

EXTI line 22 connected to COMP2 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_LINE_COMP3

EXTI line 29 connected to COMP3 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_LINE_COMP4

EXTI line 30 connected to COMP4 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_LINE_COMP5

EXTI line 31 connected to COMP5 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_LINE_COMP6

EXTI line 32 connected to COMP6 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_LINE_COMP7

EXTI line 33 connected to COMP7 output. Note: For COMPx instance availability, please refer to datasheet

COMP_EXTI_IT

EXTI line event with interruption

COMP_EXTI_EVENT

EXTI line event only (without interruption)

COMP_EXTI_RISING

EXTI line event on rising edge

COMP_EXTI_FALLING

EXTI line event on falling edge

COMP external interrupt line management

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_IT

Description:

- Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_IT

Description:

- Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP1 EXTI line.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_EVENT

Description:

- Enable the COMP1 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_EVENT

Description:

- Disable the COMP1 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GET_FLAG

Description:

- Check whether the COMP1 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP1_EXTI_CLEAR_FLAG

Description:

- Clear the COMP1 EXTI flag.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP2 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_IT

Description:

- Enable the COMP2 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_IT

Description:

- Disable the COMP2 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_EVENT

Description:

- Enable the COMP2 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_EVENT

Description:

- Disable the COMP2 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GET_FLAG

Description:

- Check whether the COMP2 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP2_EXTI_CLEAR_FLAG

Description:

- Clear the COMP2 EXTI flag.

Return value:

- None

__HAL_COMP_COMP3_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP3 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP3_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP3 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP3_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP3 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP3_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP3 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP3_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP3 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP3_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP3 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP3_EXTI_ENABLE_IT

Description:

- Enable the COMP3 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP3_EXTI_DISABLE_IT

Description:

- Disable the COMP3 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP3_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP3 EXTI line.

Return value:

- None

__HAL_COMP_COMP3_EXTI_ENABLE_EVENT

Description:

- Enable the COMP3 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP3_EXTI_DISABLE_EVENT

Description:

- Disable the COMP3 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP3_EXTI_GET_FLAG

Description:

- Check whether the COMP3 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP3_EXTI_CLEAR_FLAG

Description:

- Clear the COMP3 EXTI flag.

Return value:

- None

__HAL_COMP_COMP4_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP4 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP4_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP4 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP4_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP4 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP4_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP4 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP4_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP4 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP4_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP4 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP4_EXTI_ENABLE_IT

Description:

- Enable the COMP4 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP4_EXTI_DISABLE_IT

Description:

- Disable the COMP4 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP4_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP4 EXTI line.

Return value:

- None

__HAL_COMP_COMP4_EXTI_ENABLE_EVENT

Description:

- Enable the COMP4 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP4_EXTI_DISABLE_EVENT

Description:

- Disable the COMP4 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP4_EXTI_GET_FLAG

Description:

- Check whether the COMP4 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP4_EXTI_CLEAR_FLAG

Description:

- Clear the COMP4 EXTI flag.

Return value:

- None

__HAL_COMP_COMP5_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP5 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP5_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP5 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP5_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP5 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP5_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP5 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP5_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP5 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP5_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP5 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP5_EXTI_ENABLE_IT

Description:

- Enable the COMP5 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP5_EXTI_DISABLE_IT

Description:

- Disable the COMP5 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP5_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP5 EXTI line.

Return value:

- None

__HAL_COMP_COMP5_EXTI_ENABLE_EVENT

Description:

- Enable the COMP5 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP5_EXTI_DISABLE_EVENT

Description:

- Disable the COMP5 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP5_EXTI_GET_FLAG

Description:

- Check whether the COMP5 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP5_EXTI_CLEAR_FLAG

Description:

- Clear the COMP5 EXTI flag.

Return value:

- None

__HAL_COMP_COMP6_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP6 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP6_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP6 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP6_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP6 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP6_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP6 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP6_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP6 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP6_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP6 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP6_EXTI_ENABLE_IT

Description:

- Enable the COMP6 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP6_EXTI_DISABLE_IT

Description:

- Disable the COMP6 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP6_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP6 EXTI line.

Return value:

- None

__HAL_COMP_COMP6_EXTI_ENABLE_EVENT

Description:

- Enable the COMP6 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP6_EXTI_DISABLE_EVENT

Description:

- Disable the COMP6 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP6_EXTI_GET_FLAG

Description:

- Check whether the COMP6 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP6_EXTI_CLEAR_FLAG

Description:

- Clear the COMP6 EXTI flag.

Return value:

- None

__HAL_COMP_COMP7_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP7 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP7_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP7 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP7_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP7 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP7_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP7 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP7_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP7 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP7_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP7 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP7_EXTI_ENABLE_IT

Description:

- Enable the COMP7 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP7_EXTI_DISABLE_IT

Description:

- Disable the COMP7 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP7_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP7 EXTI line.

Return value:

- None

__HAL_COMP_COMP7_EXTI_ENABLE_EVENT

Description:

- Enable the COMP7 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP7_EXTI_DISABLE_EVENT

Description:

- Disable the COMP7 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP7_EXTI_GET_FLAG

Description:

- Check whether the COMP7 EXTI line flag is set.

Return value:

- RESET: or SET

__HAL_COMP_COMP7_EXTI_CLEAR_FLAG

Description:

- Clear the COMP7 EXTI flag.

Return value:

- None

COMP output to EXTI

COMP_TRIGGERMODE_NONE

Comparator output triggering no External Interrupt Line

COMP_TRIGGERMODE_IT_RISING

Comparator output triggering External Interrupt Line event with interruption, on rising edge

COMP_TRIGGERMODE_IT_FALLING

Comparator output triggering External Interrupt Line event with interruption, on falling edge

COMP_TRIGGERMODE_IT_RISING_FALLING

Comparator output triggering External Interrupt Line event with interruption, on both rising and falling edges

COMP_TRIGGERMODE_EVENT_RISING

Comparator output triggering External Interrupt Line event only (without interruption), on rising edge

COMP_TRIGGERMODE_EVENT_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on falling edge

COMP_TRIGGERMODE_EVENT_RISING_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on both rising and falling edges

COMP private macros to get EXTI line associated with comparators

COMP_GET_EXTI_LINE

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Handle Management

__HAL_COMP_RESET_HANDLE_STATE

Description:

- Reset COMP handle state.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

COMP_CLEAR_ERRORCODE

Description:

- Clear COMP error code (set it to no error code "HAL_COMP_ERROR_NONE").

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_ENABLE

Description:

- Enable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_DISABLE

Description:

- Disable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_LOCK

Description:

- Lock the specified comparator configuration.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

Notes:

- Using this macro induce HAL COMP handle state machine being no more in line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL_COMP_Lock')".

__HAL_COMP_IS_LOCKED

Description:

- Check whether the specified comparator is locked.

Parameters:

- __HANDLE__: COMP handle

Return value:

- Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

COMP hysteresis

COMP_HYSTERESIS_NONE

No hysteresis

COMP_HYSTERESIS_10MV

Hysteresis level 10mV

COMP_HYSTERESIS_20MV

Hysteresis level 20mV

COMP_HYSTERESIS_30MV

Hysteresis level 30mV

COMP_HYSTERESIS_40MV

Hysteresis level 40mV

COMP_HYSTERESIS_50MV

Hysteresis level 50mV

COMP_HYSTERESIS_60MV

Hysteresis level 60mV

COMP_HYSTERESIS_70MV

Hysteresis level 70mV

COMP_HYSTERESIS_LOW

Hysteresis level low

COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

COMP_HYSTERESIS_HIGH

Hysteresis level high

COMP input minus (inverting input)

COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 Channel 1 for COMP1/3/4. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 Channel 2 for COMP2/5. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_DAC2_CH1

Comparator input minus connected to DAC2 Channel 1 for COMP6/7. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_DAC3_CH1

Comparator input minus connected to DAC3 Channel 1 for COMP1/3. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_DAC3_CH2

Comparator input minus connected to DAC3 Channel 2 for COMP2/4. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_DAC4_CH1

Comparator input minus connected to DAC4 Channel 1 for COMP5/7. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_DAC4_CH2

Comparator input minus connected to DAC4 Channel 2 for COMP6. Note: For COMPx & DACx instances availability, please refer to datasheet

COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PA4 for COMP1, pin PA5 for COMP2, pin PF1 for COMP3, pin PE8 for COMP4, pin PB10 for COMP5, pin PD10 for COMP6, pin PD15 for COMP7). Note: For COMPx instance availability, please refer to datasheet

COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PA0 for COMP1, pin PA2 for COMP2, pin PC0 for COMP3, pin PB2 for COMP4, pin PD13 for COMP5, pin PB15 for COMP6, pin PB12 for COMP7). Note: For COMPx instance availability, please refer to datasheet

COMP input plus (non-inverting input)

COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA7 for COMP2, pin PA0 for COMP3, pin PB0 for COMP4, pin PB13 for COMP5, pin PB11 for COMP6, pin PB14 for COMP7). Note: For COMPx instance availability, please refer to datasheet

COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB1 for COMP1, pin PA3 for COMP2, pin PC1 for COMP3, pin PE7 for COMP4, pin PD12 for COMP5, pin PD11 for COMP6, pin PD14 for COMP7). Note: For COMPx instance availability, please refer to datasheet

COMP Output Level

COMP_OUTPUT_LEVEL_LOW

COMP_OUTPUT_LEVEL_HIGH

COMP output Polarity

COMP_OUTPUTPOL_NONINVERTED

COMP output level is not inverted (comparator output is high when the input plus is at a higher voltage than the input minus)

COMP_OUTPUTPOL_INVERTED

COMP output level is inverted (comparator output is low when the input plus is at a higher voltage than the input minus)

10 HAL CORDIC Generic Driver

10.1 CORDIC Firmware driver registers structures

10.1.1 CORDIC_HandleTypeDef

CORDIC_HandleTypeDef is defined in the stm32g4xx_hal_cordic.h

Data Fields

- ***CORDIC_TypeDef * Instance***
- ***int32_t * pInBuff***
- ***int32_t * pOutBuff***
- ***uint32_t NbCalcToOrder***
- ***uint32_t NbCalcToGet***
- ***uint32_t DMADirection***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CORDIC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***CORDIC_TypeDef* CORDIC_HandleTypeDef::Instance***
  Register base address
- ***int32_t* CORDIC_HandleTypeDef::pInBuff***
  Pointer to CORDIC input data buffer
- ***int32_t* CORDIC_HandleTypeDef::pOutBuff***
  Pointer to CORDIC output data buffer
- ***uint32_t CORDIC_HandleTypeDef::NbCalcToOrder***
  Remaining number of calculation to order
- ***uint32_t CORDIC_HandleTypeDef::NbCalcToGet***
  Remaining number of calculation result to get
- ***uint32_t CORDIC_HandleTypeDef::DMADirection***
  Direction of CORDIC DMA transfers
- ***DMA_HandleTypeDef* CORDIC_HandleTypeDef::hdmaIn***
  CORDIC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* CORDIC_HandleTypeDef::hdmaOut***
  CORDIC peripheral output data DMA handle parameters
- ***HAL_LockTypeDef CORDIC_HandleTypeDef::Lock***
  CORDIC locking object
- ***__IO HAL_CORDIC_StateTypeDef CORDIC_HandleTypeDef::State***
  CORDIC state
- ***__IO uint32_t CORDIC_HandleTypeDef::ErrorCode***
  CORDIC peripheral error code This parameter can be a value of ***CORDIC_Error_Code***

10.1.2 CORDIC_ConfigTypeDef

CORDIC_ConfigTypeDef is defined in the stm32g4xx_hal_cordic.h

Data Fields

- ***uint32_t Function***
- ***uint32_t Scale***

- ***uint32_t InSize***
- ***uint32_t OutSize***
- ***uint32_t NbWrite***
- ***uint32_t NbRead***
- ***uint32_t Precision***

Field Documentation

- ***uint32_t CORDIC_ConfigTypeDef::Function***
  Function This parameter can be a value of ***CORDIC_Function***
- ***uint32_t CORDIC_ConfigTypeDef::Scale***
  Scaling factor This parameter can be a value of ***CORDIC_Scale***
- ***uint32_t CORDIC_ConfigTypeDef::InSize***
  Width of input data This parameter can be a value of ***CORDIC_In_Size***
- ***uint32_t CORDIC_ConfigTypeDef::OutSize***
  Width of output data This parameter can be a value of ***CORDIC_Out_Size***
- ***uint32_t CORDIC_ConfigTypeDef::NbWrite***
  Number of 32-bit write expected for one calculation This parameter can be a value of ***CORDIC_Nb_Write***
- ***uint32_t CORDIC_ConfigTypeDef::NbRead***
  Number of 32-bit read expected after one calculation This parameter can be a value of ***CORDIC_Nb_Read***
- ***uint32_t CORDIC_ConfigTypeDef::Precision***
  Number of cycles for calculation This parameter can be a value of ***CORDIC_Precision_In_Cycles_Number***

10.2 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

10.2.1 How to use this driver

The CORDIC HAL driver can be used as follows:

1. Initialize the CORDIC low level resources by implementing the HAL_CORDIC_MspInit():
   – Enable the CORDIC interface clock using __HAL_RCC_CORDIC_CLK_ENABLE()
   – In case of using interrupts (e.g. HAL_CORDIC_Calculate_IT())
     ◦ Configure the CORDIC interrupt priority using HAL_NVIC_SetPriority()
     ◦ Enable the CORDIC IRQ handler using HAL_NVIC_EnableIRQ()
     ◦ In CORDIC IRQ handler, call HAL_CORDIC_IRQHandler()
   – In case of using DMA to control data transfer (e.g. HAL_CORDIC_Calculate_DMA())
     ◦ Enable the DMA2 interface clock using __HAL_RCC_DMA2_CLK_ENABLE()
     ◦ Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
     ◦ Associate the initialized DMA handle to the CORDIC DMA handle using __HAL_LINKDMA()
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. Resort to HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the CORDIC HAL using HAL_CORDIC_Init(). This function
   – resorts to HAL_CORDIC_MspInit() for low-level initialization,

3. Configure CORDIC processing (calculation) using HAL_CORDIC_Configure(). This function configures:
   – Processing functions: Cosine, Sine, Phase, Modulus, Arctangent, Hyperbolic cosine, Hyperbolic sine, Hyperbolic arctangent, Natural log, Square root
   – Scaling factor: 1 to 2exp(-7)
   – Width of input data: 32 bits input data size (Q1.31 format) or 16 bits input data size (Q1.15 format)
   – Width of output data: 32 bits output data size (Q1.31 format) or 16 bits output data size (Q1.15 format)
   – Number of 32-bit write expected for one calculation: One 32-bits write or Two 32-bit write
   – Number of 32-bit read expected after one calculation: One 32-bits read or Two 32-bit read
   – Precision: 1 to 15 cycles for calculation (the more cycles, the better precision)
4. Four processing (calculation) functions are available:
   – Polling mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished API is HAL_CORDIC_Calculate
   – Polling Zero-overhead mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished A bit faster than standard polling mode, but blocking also AHB bus API is HAL_CORDIC_CalculateZO
   – Interrupt mode: processing API is not blocking functions i.e. it processes the data under interrupt API is HAL_CORDIC_Calculate_IT
   – DMA mode: processing API is not blocking functions and the CPU is not used for data transfer, i.e. the data transfer is ensured by DMA API is HAL_CORDIC_Calculate_DMA
5. Call HAL_CORDIC_DeInit() to de-initialize the CORDIC peripheral. This function
   – resorts to HAL_CORDIC_MspDeInit() for low-level de-initialization,

Callback registration

10.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CORDIC peripheral and the associated handle
- DeInitialize the CORDIC peripheral
- Initialize the CORDIC MSP (MCU Specific Package)
- De-Initialize the CORDIC MSP

This section contains the following APIs:

- ***HAL_CORDIC_Init***
- ***HAL_CORDIC_DeInit***
- ***HAL_CORDIC_MspInit***
- ***HAL_CORDIC_MspDeInit***
- ***HAL_CORDIC_Configure***
- ***HAL_CORDIC_Calculate***
- ***HAL_CORDIC_CalculateZO***
- ***HAL_CORDIC_Calculate_IT***
- ***HAL_CORDIC_Calculate_DMA***

10.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure the CORDIC peripheral: function, precision, scaling factor, number of input data and output data, size of input data and output data.
- Calculate output data of CORDIC processing on input date, using the existing CORDIC configuration

Four processing functions are available for calculation:

- Polling mode
- Polling mode, with Zero-Overhead register access
- Interrupt mode
- DMA mode

This section contains the following APIs:

- *HAL_CORDIC_Configure*
- *HAL_CORDIC_Calculate*
- *HAL_CORDIC_CalculateZO*
- *HAL_CORDIC_Calculate_IT*
- *HAL_CORDIC_Calculate_DMA*

10.2.4 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt calculate complete
- DMA or Interrupt error

This section contains the following APIs:

- *HAL_CORDIC_ErrorCallback*
- *HAL_CORDIC_CalculateCpltCallback*

10.2.5 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_CORDIC_IRQHandler*

10.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_CORDIC_GetState*
- *HAL_CORDIC_GetError*

10.2.7 Detailed description of functions

HAL_CORDIC_Init

Function name

HAL_StatusTypeDef HAL_CORDIC_Init (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC peripheral and the associated handle.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_DeInit

Function name

HAL_StatusTypeDef HAL_CORDIC_DeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC peripheral.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_MspInit

Function name

void HAL_CORDIC_MspInit (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_MspDeInit

Function name

void HAL_CORDIC_MspDeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_Configure

Function name

HAL_StatusTypeDef HAL_CORDIC_Configure (CORDIC_HandleTypeDef * hcordic, CORDIC_ConfigTypeDef * sConfig)

Function description

Configure the CORDIC processing according to the specified parameters in the CORDIC_ConfigTypeDef structure.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **sConfig:** pointer to a CORDIC_ConfigTypeDef structure that contains the CORDIC configuration information.

Return values

- **HAL:** status

HAL_CORDIC_Calculate

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate (CORDIC_HandleTypeDef * hcordic, int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in polling mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_CalculateZO

Function name

HAL_StatusTypeDef HAL_CORDIC_CalculateZO (CORDIC_HandleTypeDef * hcordic, int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in Zero-Overhead mode (output data being read soon as input data are written), according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_Calculate_IT

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_IT (CORDIC_HandleTypeDef * hcordic, int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc)

Function description

Carry out data of CORDIC processing in interrupt mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.

Return values

- **HAL:** status

HAL_CORDIC_Calculate_DMA

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_DMA (CORDIC_HandleTypeDef * hcordic, int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t DMADirection)

Function description

Carry out input and/or output data of CORDIC processing in DMA mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **DMADirection:** Direction of DMA transfers. This parameter can be one of the following values:
  – CORDIC DMA direction CORDIC DMA direction

Return values

- **HAL:** status

Notes

- pInBuff or pOutBuff is unused in case of unique DMADirection transfer, and can be set to NULL value in this case.
- pInBuff and pOutBuff buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the Peripheral.

HAL_CORDIC_ErrorCallback

Function name

void HAL_CORDIC_ErrorCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC error callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_CalculateCpltCallback

Function name

void HAL_CORDIC_CalculateCpltCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC calculate complete callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_IRQHandler

Function name

void HAL_CORDIC_IRQHandler (CORDIC_HandleTypeDef * hcordic)

Function description

Handle CORDIC interrupt request.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_GetState

Function name

HAL_CORDIC_StateTypeDef HAL_CORDIC_GetState (CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC handle state.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **HAL:** state

HAL_CORDIC_GetError

Function name

uint32_t HAL_CORDIC_GetError (CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC peripheral error.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **Error:** bit-map

Notes

- The returned error is a bit-map combination of possible errors

10.3 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

10.3.1 CORDIC

CORDIC

DMA Read Request Enable bit

CORDIC_DMA_REN

DMA Read requests enable

DMA Write Request Enable bit

CORDIC_DMA_WEN

DMA Write channel enable

CORDIC DMA direction

CORDIC_DMA_DIR_NONE

DMA direction : none

CORDIC_DMA_DIR_IN

DMA direction : Input of CORDIC

CORDIC_DMA_DIR_OUT

DMA direction : Output of CORDIC

CORDIC_DMA_DIR_IN_OUT

DMA direction : Input and Output of CORDIC

CORDIC Error code

HAL_CORDIC_ERROR_NONE

No error

HAL_CORDIC_ERROR_PARAM

Wrong parameter error

HAL_CORDIC_ERROR_NOT_READY

Peripheral not ready

HAL_CORDIC_ERROR_TIMEOUT

Timeout error

HAL_CORDIC_ERROR_DMA

DMA error

CORDIC Exported Macros

__HAL_CORDIC_RESET_HANDLE_STATE

Description:

- Reset CORDIC handle state.

Parameters:

- __HANDLE__: CORDIC handle

Return value:

- None

__HAL_CORDIC_ENABLE_IT

Description:

- Enable the CORDIC interrupt when result is ready.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_DISABLE_IT

Description:

- Disable the CORDIC interrupt.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_GET_IT

Description:

- Check whether the specified CORDIC interrupt occurred or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_CORDIC_CLEAR_IT

Description:

- Clear specified CORDIC interrupt status.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to clear

Return value:

- None

__HAL_CORDIC_GET_FLAG

Description:

- Check whether the specified CORDIC status flag is set or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to check This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_CORDIC_CLEAR_FLAG

Description:

- Clear specified CORDIC status flag.

Parameters:

- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to clear This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:

- None

__HAL_CORDIC_GET_IT_SOURCE

Description:

- Check whether the specified CORDIC interrupt is enabled or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- FlagStatus

CORDIC status flags

CORDIC_FLAG_RRDY

Result Ready Flag

CORDIC Function

CORDIC_FUNCTION_COSINE

Cosine

CORDIC_FUNCTION_SINE

Sine

CORDIC_FUNCTION_PHASE

Phase

CORDIC_FUNCTION_MODULUS

Modulus

CORDIC_FUNCTION_ARCTANGENT

Arctangent

CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

CORDIC_FUNCTION_HSINE

Hyperbolic Sine

CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

CORDIC_FUNCTION_SQUAREROOT

Square Root

CORDIC Interrupts Enable bit

CORDIC_IT_IEN

Result ready interrupt enable

CORDIC input data size

CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

CORDIC Number of 32-bit read required after one calculation

CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data ouput (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

CORDIC Number of 32-bit write required for one calculation

CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

CORDIC Results Size

CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

CORDIC Precision in Cycles Number

CORDIC_PRECISION_1CYCLE

CORDIC_PRECISION_2CYCLES

CORDIC_PRECISION_3CYCLES

CORDIC_PRECISION_4CYCLES

CORDIC_PRECISION_5CYCLES

CORDIC_PRECISION_6CYCLES

CORDIC_PRECISION_7CYCLES

CORDIC_PRECISION_8CYCLES

CORDIC_PRECISION_9CYCLES

CORDIC_PRECISION_10CYCLES

CORDIC_PRECISION_11CYCLES

CORDIC_PRECISION_12CYCLES

CORDIC_PRECISION_13CYCLES

CORDIC_PRECISION_14CYCLES

CORDIC_PRECISION_15CYCLES

CORDIC Scaling factor

CORDIC_SCALE_0

CORDIC_SCALE_1

CORDIC_SCALE_2

CORDIC_SCALE_3

CORDIC_SCALE_4

CORDIC_SCALE_5

CORDIC_SCALE_6

CORDIC_SCALE_7

11 HAL CORTEX Generic Driver

11.1 CORTEX Firmware driver registers structures

11.1.1 MPU_Region_InitTypeDef

MPU_Region_InitTypeDef is defined in the stm32g4xx_hal_cortex.h

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint8_t Size***
- ***uint8_t SubRegionDisable***
- ***uint8_t TypeExtField***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t IsShareable***
- ***uint8_t IsCacheable***
- ***uint8_t IsBufferable***

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the number of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::Size***
  Specifies the size of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Size***
- ***uint8_t MPU_Region_InitTypeDef::SubRegionDisable***
  Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint8_t MPU_Region_InitTypeDef::TypeExtField***
  Specifies the TEX field level. This parameter can be a value of ***CORTEX_MPU_TEX_Levels***
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***
- ***uint8_t MPU_Region_InitTypeDef::IsCacheable***
  Specifies the cacheable status of the region protected. This parameter can be a value of ***CORTEX_MPU_Access_Cacheable***
- ***uint8_t MPU_Region_InitTypeDef::IsBufferable***
  Specifies the bufferable status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Bufferable***

11.2 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

11.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M4 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ().

Note: When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority.

Note: IRQ priority order (sorted by highest to lowest priority):

- *Lowest pre-emption priority*
- *Lowest sub priority*
- *Lowest hardware priority (IRQ number)*

How to configure SysTick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro __HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The __HAL_CORTEX_SYSTICKCLK_CONFIG() macro is defined inside the stm32g4xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

11.2.2 Initialization and Configuration functions

This section provides the CORTEX HAL driver functions allowing to configure Interrupts SysTick functionalities

This section contains the following APIs:

- ***HAL_NVIC_SetPriorityGrouping***
- ***HAL_NVIC_SetPriority***
- ***HAL_NVIC_EnableIRQ***
- ***HAL_NVIC_DisableIRQ***
- ***HAL_NVIC_SystemReset***
- ***HAL_SYSTICK_Config***

11.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK, MPU) functionalities.

This section contains the following APIs:

- *HAL_NVIC_GetPriorityGrouping*
- *HAL_NVIC_GetPriority*
- *HAL_NVIC_SetPendingIRQ*
- *HAL_NVIC_GetPendingIRQ*
- *HAL_NVIC_ClearPendingIRQ*
- *HAL_NVIC_GetActive*
- *HAL_SYSTICK_CLKSourceConfig*
- *HAL_SYSTICK_IRQHandler*
- *HAL_SYSTICK_Callback*
- *HAL_MPU_Enable*
- *HAL_MPU_Disable*
- *HAL_MPU_ConfigRegion*

11.2.4 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name

void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)

Function description

Set the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.

Parameters

- **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority

Return values

- **None:**

Notes

- When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name

void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)

Function description

Set the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))
- **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.

Return values

- **None:**

HAL_NVIC_EnableIRQ

Function name

void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)

Function description

Enable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name

void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)

Function description

Disable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))

Return values

- **None:**

HAL_NVIC_SystemReset

Function name

void HAL_NVIC_SystemReset (void)

Function description

Initiate a system reset request to reset the MCU.

Return values

- **None:**

HAL_SYSTICK_Config

Function name

uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)

Function description

Initialize the System Timer with interrupt enabled and start the System Tick Timer (SysTick): Counter is in free running mode to generate periodic interrupts.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- **status:** - 0 Function succeeded.
  - 1 Function failed.

HAL_NVIC_GetPriorityGrouping

Function name

uint32_t HAL_NVIC_GetPriorityGrouping (void)

Function description

Get the priority grouping field from the NVIC Interrupt Controller.

Return values

- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name

void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)

Function description

Get the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- **None:**

HAL_NVIC_GetPendingIRQ

Function name

uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)

Function description

Get Pending Interrupt (read the pending register in the NVIC and return the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name

void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)

Function description

Set Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))

Return values

- **None:**

HAL_NVIC_ClearPendingIRQ

Function name

void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

Function description

Clear the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))

Return values

- **None:**

HAL_NVIC_GetActive

Function name

uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)

Function description

Get active interrupt (read the active register in NVIC and return the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32g4xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name

void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

Function description

Configure the SysTick clock source.

Parameters

- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
  - SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return values

- **None:**

HAL_SYSTICK_IRQHandler

Function name

void HAL_SYSTICK_IRQHandler (void)

Function description

Handle SYSTICK interrupt request.

Return values

- **None:**

HAL_SYSTICK_Callback

Function name

void HAL_SYSTICK_Callback (void)

Function description

SYSTICK callback.

Return values

- **None:**

HAL_MPU_Enable

Function name

void HAL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable the MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged accessto the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values

- **None:**

HAL_MPU_Disable

Function name

void HAL_MPU_Disable (void)

Function description

Disable the MPU.

Return values

- **None:**

HAL_MPU_ConfigRegion

Function name

void HAL_MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)

Function description

Initialize and configure the Region and the memory to be protected.

Parameters

- **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

11.3 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

11.3.1 CORTEX

CORTEX

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE

MPU_ACCESS_NOT_SHAREABLE

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B

MPU_REGION_SIZE_64B

MPU_REGION_SIZE_128B

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB

MPU_REGION_SIZE_256KB

MPU_REGION_SIZE_512KB

MPU_REGION_SIZE_1MB

MPU_REGION_SIZE_2MB

MPU_REGION_SIZE_4MB

MPU_REGION_SIZE_8MB

MPU_REGION_SIZE_16MB

MPU_REGION_SIZE_32MB

MPU_REGION_SIZE_64MB

MPU_REGION_SIZE_128MB

MPU_REGION_SIZE_256MB

MPU_REGION_SIZE_512MB

MPU_REGION_SIZE_1GB

MPU_REGION_SIZE_2GB

MPU_REGION_SIZE_4GB

CORTEX MPU TEX Levels

MPU_TEX_LEVEL0

MPU_TEX_LEVEL1

MPU_TEX_LEVEL2

MPU_TEX_LEVEL4

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0

0 bit for pre-emption priority, 4 bits for subpriority

NVIC_PRIORITYGROUP_1

1 bit for pre-emption priority, 3 bits for subpriority

NVIC_PRIORITYGROUP_2

2 bits for pre-emption priority, 2 bits for subpriority

NVIC_PRIORITYGROUP_3

3 bits for pre-emption priority, 1 bit for subpriority

NVIC_PRIORITYGROUP_4

4 bits for pre-emption priority, 0 bit for subpriority

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

12 HAL CRC Generic Driver

12.1 CRC Firmware driver registers structures

12.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32g4xx_hal_crc.h

Data Fields

- ***uint8_t DefaultPolynomialUse***
- ***uint8_t DefaultInitValueUse***
- ***uint32_t GeneratingPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t InitValue***
- ***uint32_t InputDataInversionMode***
- ***uint32_t OutputDataInversionMode***

Field Documentation

- ***uint8_t CRC_InitTypeDef::DefaultPolynomialUse***

  This parameter is a value of ***CRC_Default_Polynomial*** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set.

- ***uint8_t CRC_InitTypeDef::DefaultInitValueUse***

  This parameter is a value of ***CRC_Default_InitValue_Use*** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set.

- ***uint32_t CRC_InitTypeDef::GeneratingPolynomial***

  Set CRC generating polynomial as a 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal representation, e.g., for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE.

- ***uint32_t CRC_InitTypeDef::CRCLength***

  This parameter is a value of ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of
  - **CRC_POLYLENGTH_32B** (32-bit CRC),
  - **CRC_POLYLENGTH_16B** (16-bit CRC),
  - **CRC_POLYLENGTH_8B** (8-bit CRC),
  - **CRC_POLYLENGTH_7B** (7-bit CRC).

- ***uint32_t CRC_InitTypeDef::InitValue***

  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE.

- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***

  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values
  - **CRC_INPUTDATA_INVERSION_NONE** no input data inversion
  - **CRC_INPUTDATA_INVERSION_BYTE** byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2
  - **CRC_INPUTDATA_INVERSION_HALFWORD** halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C
  - **CRC_INPUTDATA_INVERSION_WORD** word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***
  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output data (i.e. CRC) inversion mode. Can be either
  - **CRC_OUTPUTDATA_INVERSION_DISABLE** no CRC inversion,
  - **CRC_OUTPUTDATA_INVERSION_ENABLE** CRC 0x11223344 is converted into 0x22CC4488

12.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32g4xx_hal_crc.h

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
  Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***
  CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
  CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
  CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***
  This parameter is a value of ***CRC_Input_Buffer_Format*** and specifies input data format. Can be either
  - **CRC_INPUTDATA_FORMAT_BYTES** input data is a stream of bytes (8-bit data)
  - **CRC_INPUTDATA_FORMAT_HALFWORDS** input data is a stream of half-words (16-bit data)
  - **CRC_INPUTDATA_FORMAT_WORDS** input data is a stream of words (32-bit data)

  Note that constant CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

12.2 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

12.2.1 How to use this driver

- Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
- Initialize CRC calculator
  - specify generating polynomial (peripheral default or non-default one)
  - specify initialization value (peripheral default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
- Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
- Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

12.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle

- DeInitialize the CRC peripheral
- Initialize the CRC MSP (MCU Specific Package)
- DeInitialize the CRC MSP

This section contains the following APIs:

- ***HAL_CRC_Init***
- ***HAL_CRC_DeInit***
- ***HAL_CRC_MspInit***
- ***HAL_CRC_MspDeInit***

12.2.3 Peripheral Control functions

This section provides functions allowing to:

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using combination of the previous CRC value and the new one.

or

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- ***HAL_CRC_Accumulate***
- ***HAL_CRC_Calculate***

12.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- ***HAL_CRC_GetState***

12.2.5 Detailed description of functions

HAL_CRC_Init

Function name

HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)

Function description

Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_DeInit

Function name

HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC peripheral.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_MspInit

Function name

void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)

Function description

Initializes the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_MspDeInit

Function name

void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_Accumulate

Function name

uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_Calculate

Function name

uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_GetState

Function name

HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)

Function description

Return the CRC handle state.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** state

12.3 CRC Firmware driver defines

The following section lists the various define and macros of the module.

12.3.1 CRC

CRC

CRC API aliases

HAL_CRC_Input_Data_Reverse

Aliased to HAL_CRCEx_Input_Data_Reverse for inter STM32 series compatibility

HAL_CRC_Output_Data_Reverse

Aliased to HAL_CRCEx_Output_Data_Reverse for inter STM32 series compatibility

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Initial CRC default value

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

Enable initial CRC default value

DEFAULT_INIT_VALUE_DISABLE

Disable initial CRC default value

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

Enable default generating polynomial 0x04C11DB7

DEFAULT_POLYNOMIAL_DISABLE

Disable default generating polynomial 0x04C11DB7

Default CRC generating polynomial

DEFAULT_CRC32_POLY

X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE

Description:

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

- __HANDLE__: CRC handle
- __INIT__: 32-bit initial value

Return value:

- None

__HAL_CRC_SET_IDR

Description:

- Store data in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: Value to be stored in the ID register

Return value:

- None

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

__HAL_CRC_GET_IDR

Description:

- Return the data stored in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- Value: of the ID register

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

Undefined input data format

CRC_INPUTDATA_FORMAT_BYTES

Input data in byte format

CRC_INPUTDATA_FORMAT_HALFWORDS

Input data in half-word format

CRC_INPUTDATA_FORMAT_WORDS

Input data in word format

Polynomial sizes to configure the peripheral

CRC_POLYLENGTH_32B

Resort to a 32-bit long generating polynomial

CRC_POLYLENGTH_16B

Resort to a 16-bit long generating polynomial

CRC_POLYLENGTH_8B

Resort to a 8-bit long generating polynomial

CRC_POLYLENGTH_7B

Resort to a 7-bit long generating polynomial

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

32-bit long CRC

HAL_CRC_LENGTH_16B

16-bit long CRC

HAL_CRC_LENGTH_8B

8-bit long CRC

HAL_CRC_LENGTH_7B

7-bit long CRC

13 HAL CRC Extension Driver

13.1 CRCEx Firmware driver API description

The following section lists the various functions of the CRCEx library.

13.1.1 How to use this driver

- Set user-defined generating polynomial thru HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

13.1.2 Extended configuration functions

This section provides functions allowing to:

- Configure the generating polynomial
- Configure the input data inversion
- Configure the output data inversion

This section contains the following APIs:

- ***HAL_CRCEx_Polynomial_Set***
- ***HAL_CRCEx_Input_Data_Reverse***
- ***HAL_CRCEx_Output_Data_Reverse***

13.1.3 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name

HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)

Function description

Initialize the CRC polynomial if different from default one.

Parameters

- **hcrc:** CRC handle
- **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long). This parameter is written in normal representation, e.g.
  - for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
  - for a polynomial of degree 16, X^16 + X^12 + X^5 + 1 is written 0x1021
- **PolyLength:** CRC polynomial length. This parameter can be one of the following values:
  - CRC_POLYLENGTH_7B 7-bit long CRC (generating polynomial of degree 7)
  - CRC_POLYLENGTH_8B 8-bit long CRC (generating polynomial of degree 8)
  - CRC_POLYLENGTH_16B 16-bit long CRC (generating polynomial of degree 16)
  - CRC_POLYLENGTH_32B 32-bit long CRC (generating polynomial of degree 32)

Return values

- **HAL:** status

HAL_CRCEx_Input_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)

Function description

Set the Reverse Input data mode.

Parameters

- **hcrc:** CRC handle
- **InputReverseMode:** Input Data inversion mode. This parameter can be one of the following values:
  - CRC_INPUTDATA_INVERSION_NONE no change in bit order (default value)
  - CRC_INPUTDATA_INVERSION_BYTE Byte-wise bit reversal
  - CRC_INPUTDATA_INVERSION_HALFWORD HalfWord-wise bit reversal
  - CRC_INPUTDATA_INVERSION_WORD Word-wise bit reversal

Return values

- **HAL:** status

HAL_CRCEx_Output_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)

Function description

Set the Reverse Output data mode.

Parameters

- **hcrc:** CRC handle
- **OutputReverseMode:** Output Data inversion mode. This parameter can be one of the following values:
  - CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion (default value)
  - CRC_OUTPUTDATA_INVERSION_ENABLE bit-level inversion (e.g. for a 8-bit CRC: 0xB5 becomes 0xAD)

Return values

- **HAL:** status

13.2 CRCEx Firmware driver defines

The following section lists the various define and macros of the module.

13.2.1 CRCEx

CRCEx

CRC Extended Exported Macros

__HAL_CRC_OUTPUTREVERSAL_ENABLE

Description:

- Set CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_DISABLE

Description:

- Unset CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_POLYNOMIAL_CONFIG

Description:

- Set CRC non-default polynomial.

Parameters:

- __HANDLE__: CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:

- None

Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE

No input data inversion

CRC_INPUTDATA_INVERSION_BYTE

Byte-wise input data inversion

CRC_INPUTDATA_INVERSION_HALFWORD

HalfWord-wise input data inversion

CRC_INPUTDATA_INVERSION_WORD

Word-wise input data inversion

Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLE

No output data inversion

CRC_OUTPUTDATA_INVERSION_ENABLE

Bit-wise output data inversion

14 HAL CRYP Generic Driver

14.1 CRYP Firmware driver registers structures

14.1.1 CRYP_ConfigTypeDef

CRYP_ConfigTypeDef is defined in the stm32g4xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t * Header***
- ***uint32_t HeaderSize***
- ***uint32_t * B0***
- ***uint32_t DataWidthUnit***
- ***uint32_t HeaderWidthUnit***
- ***uint32_t KeyIVConfigSkip***

Field Documentation

- ***uint32_t CRYP_ConfigTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ConfigTypeDef::KeySize***
  Used only in AES mode : 128, 192 or 256 bit key length in CRYP1. 128 or 256 bit key length in TinyAES This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ConfigTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ConfigTypeDef::pInitVect***
  The initialization vector used also as initialization counter in CTR mode
- ***uint32_t CRYP_ConfigTypeDef::Algorithm***
  DES/ TDES Algorithm ECB/CBC AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t* CRYP_ConfigTypeDef::Header***
  used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.
- ***uint32_t CRYP_ConfigTypeDef::HeaderSize***
  The size of header buffer
- ***uint32_t* CRYP_ConfigTypeDef::B0***
  B0 is first authentication block used only in AES CCM mode
- ***uint32_t CRYP_ConfigTypeDef::DataWidthUnit***
  Payload Data Width Unit, this parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::HeaderWidthUnit***
  Header Width Unit, this parameter can be value of ***CRYP_Header_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::KeyIVConfigSkip***
  CRYP peripheral Key and IV configuration skip, to config Key and Initialization Vector only once and to skip configuration for consecutive processings. This parameter can be a value of ***CRYP_Configuration_Skip***

14.1.2 CRYP_HandleTypeDef

CRYP_HandleTypeDef is defined in the stm32g4xx_hal_cryp.h

Data Fields

- ***AES_TypeDef * Instance***
- ***CRYP_ConfigTypeDef Init***
- ***FunctionalState AutoKeyDerivation***
- ***uint32_t * pCrypInBuffPtr***
- ***uint32_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypHeaderCount***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***uint16_t Size***
- ***uint32_t Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t KeyIVConfig***
- ***uint32_t SizesSum***

Field Documentation

- ***AES_TypeDef* CRYP_HandleTypeDef::Instance***
  AES Register base address
- ***CRYP_ConfigTypeDef CRYP_HandleTypeDef::Init***
  CRYP required parameters
- ***FunctionalState CRYP_HandleTypeDef::AutoKeyDerivation***
  Used only in TinyAES to allow to bypass or not key write-up before decryption. This parameter can be a value of ENABLE/DISABLE
- ***uint32_t* CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint32_t* CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t CRYP_HandleTypeDef::CrypHeaderCount***
  Counter of header data
- ***__IO uint16_t CRYP_HandleTypeDef::CrypInCount***
  Counter of input data
- ***__IO uint16_t CRYP_HandleTypeDef::CrypOutCount***
  Counter of output data
- ***uint16_t CRYP_HandleTypeDef::Size***
  length of input data in words
- ***uint32_t CRYP_HandleTypeDef::Phase***
  CRYP peripheral phase
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmain***
  CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmaout***
  CRYP Out DMA handle parameters
- ***HAL_LockTypeDef CRYP_HandleTypeDef::Lock***
  CRYP locking object
- ***__IO HAL_CRYP_STATETypeDef CRYP_HandleTypeDef::State***
  CRYP peripheral state

- ***__IO uint32_t CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***uint32_t CRYP_HandleTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag, used when configuration can be skipped
- ***uint32_t CRYP_HandleTypeDef::SizesSum***
  Sum of successive payloads lengths (in bytes), stored for a single signature computation after several messages processing

14.2 CRYP Firmware driver API description

The following section lists the various functions of the CRYP library.

14.2.1 How to use this driver

The CRYP HAL driver can be used in CRYP or TinyAES peripheral as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
   a. Enable the CRYP interface clock using __HAL_RCC_CRYP_CLK_ENABLE()or __HAL_RCC_AES_CLK_ENABLE for TinyAES peripheral
   b. In case of using interrupts (e.g. HAL_CRYP_Encrypt_IT())
      - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
      - Enable the CRYP IRQ handler using HAL_NVIC_EnableIRQ()
      - In CRYP IRQ handler, call HAL_CRYP_IRQHandler()
   c. In case of using DMA to control data transfer (e.g. HAL_CRYP_Encrypt_DMA())
      - Enable the DMAx interface clock using __RCC_DMAx_CLK_ENABLE()
      - Configure and enable two DMA streams one for managing data transfer from memory to peripheral (input stream) and another stream for managing data transfer from peripheral to memory (output stream)
      - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the CRYP according to the specified parameters :
   a. The data type: 1-bit, 8-bit, 16-bit or 32-bit.
   b. The key size: 128, 192 or 256.
   c. The AlgoMode DES/ TDES Algorithm ECB/CBC or AES Algorithm ECB/CBC/CTR/GCM or CCM.
   d. The initialization vector (counter). It is not used in ECB mode.
   e. The key buffer used for encryption/decryption.
      - In some specific configurations, the key is written by the application code out of the HAL scope. In that case, user can still resort to the HAL APIs as usual but must make sure that pKey pointer is set to NULL.
   f. The Header used only in AES GCM and CCM Algorithm for authentication.
   g. The HeaderSize The size of header buffer in word.
   h. The B0 block is the first authentication block used only in AES CCM mode.
3. Three processing (encryption/decryption) functions are available:
   a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
   b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
   c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA
4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is configured and processes the buffer in input. At second call, no need to Initialize the CRYP, user have to get current configuration via HAL_CRYP_GetConfig() API, then only HAL_CRYP_SetConfig() is requested to set new parametres, finally user can start encryption/decryption.

5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.
6. To process a single message with consecutive calls to HAL_CRYP_Encrypt() or HAL_CRYP_Decrypt() without having to configure again the Key or the Initialization Vector between each API call, the field KeyIVConfigSkip of the initialization structure must be set to CRYP_KEYIVCONFIG_ONCE. Same is true for consecutive calls of HAL_CRYP_Encrypt_IT(), HAL_CRYP_Decrypt_IT(), HAL_CRYP_Encrypt_DMA() or HAL_CRYP_Decrypt_DMA().

The cryptographic processor supports following standards:

1. The data encryption standard (DES) and Triple-DES (TDES) supported only by CRYP1 peripheral:
   a. 64-bit data block processing
   b. chaining modes supported :
      - Electronic Code Book(ECB)
      - Cipher Block Chaining (CBC)
   c. keys length supported :64-bit, 128-bit and 192-bit.
2. The advanced encryption standard (AES) supported by CRYP1 & TinyAES peripheral:
   a. 128-bit data block processing
   b. chaining modes supported :
      - Electronic Code Book(ECB)
      - Cipher Block Chaining (CBC)
      - Counter mode (CTR)
      - Galois/counter mode (GCM/GMAC)
      - Counter with Cipher Block Chaining-Message(CCM)
   c. keys length Supported :
      - for CRYP1 peripheral: 128-bit, 192-bit and 256-bit.
      - for TinyAES peripheral: 128-bit and 256-bit

Note: Specific care must be taken to format the key and the Initialization Vector IV!

If the key is defined as a 128-bit long array key[127..0] = {b127 ... b0} where b127 is the MSB and b0 the LSB, the key must be stored in MCU memory

- as a sequence of words where the MSB word comes first (occupies the lowest memory address)
  – address n+0 : 0b b127 .. b120 b119 .. b112 b111 .. b104 b103 .. b96
  – address n+4 : 0b b95 .. b88 b87 .. b80 b79 .. b72 b71 .. b64
  – address n+8 : 0b b63 .. b56 b55 .. b48 b47 .. b40 b39 .. b32
  – address n+C : 0b b31 .. b24 b23 .. b16 b15 .. b8 b7 .. b0

Hereafter, another illustration when considering a 128-bit long key made of 16 bytes {B15..B0}. The 4 32-bit words that make the key must be stored as follows in MCU memory:

- address n+0 : 0x B15 B14 B13 B12
- address n+4 : 0x B11 B10 B9 B8
- address n+8 : 0x B7 B6 B5 B4
- address n+C : 0x B3 B2 B1 B0

which leads to the expected setting

- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Same format must be applied for a 256-bit long key made of 32 bytes {B31..B0}. The 8 32-bit words that make the key must be stored as follows in MCU memory:

- address n+00 : 0x B31 B30 B29 B28
- address n+04 : 0x B27 B26 B25 B24
- address n+08 : 0x B23 B22 B21 B20
- address n+0C : 0x B19 B18 B17 B16

- address n+10 : 0x B15 B14 B13 B12
- address n+14 : 0x B11 B10 B9 B8
- address n+18 : 0x B7 B6 B5 B4
- address n+1C : 0x B3 B2 B1 B0

which leads to the expected setting

- AES_KEYR7 = 0x B31 B30 B29 B28
- AES_KEYR6 = 0x B27 B26 B25 B24
- AES_KEYR5 = 0x B23 B22 B21 B20
- AES_KEYR4 = 0x B19 B18 B17 B16
- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Initialization Vector IV (4 32-bit words) format must follow the same as that of a 128-bit long key.

Note that key and IV registers are not sensitive to swap mode selection.

This section describes the AES Galois/counter mode (GCM) supported by both CRYP1 and TinyAES peripherals:

1. Algorithm supported :
   a. Galois/counter mode (GCM)
   b. Galois message authentication code (GMAC) :is exactly the same as GCM algorithm composed only by an header.
2. Four phases are performed in GCM :
   a. Init phase: peripheral prepares the GCM hash subkey (H) and do the IV processing
   b. Header phase: peripheral processes the Additional Authenticated Data (AAD), with hash computation only.
   c. Payload phase: peripheral processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
   d. Final phase: peripheral generates the authenticated tag (T) using the last block of data.
3. structure of message construction in GCM is defined as below :
   a. 16 bytes Initial Counter Block (ICB)composed of IV and counter
   b. The authenticated header A (also knows as Additional Authentication Data AAD) this part of the message is only authenticated, not encrypted.
   c. The plaintext message P is both authenticated and encrypted as ciphertext. GCM standard specifies that ciphertext has same bit length as the plaintext.
   d. The last block is composed of the length of A (on 64 bits) and the length of ciphertext (on 64 bits)

A more detailed description of the GCM message structure is available below.

This section describe The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) supported by both CRYP1 and TinyAES peripheral:

1. Specific parameters for CCM :
   a. B0 block : follows NIST Special Publication 800-38C,
   b. B1 block (header)
   c. CTRx block : control blocks

A detailed description of the CCM message structure is available below.

1. Four phases are performed in CCM for CRYP1 peripheral:
   a. Init phase: peripheral prepares the GCM hash subkey (H) and do the IV processing
   b. Header phase: peripheral processes the Additional Authenticated Data (AAD), with hash computation only.
   c. Payload phase: peripheral processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
   d. Final phase: peripheral generates the authenticated tag (T) using the last block of data.

2. CCM in TinyAES peripheral:
   a. To perform message payload encryption or decryption AES is configured in CTR mode.
   b. For authentication two phases are performed : - Header phase: peripheral processes the Additional Authenticated Data (AAD) first, then the cleartext message only cleartext payload (not the ciphertext payload) is used and no outpout.
   c. Final phase: peripheral generates the authenticated tag (T) using the last block of data.

Callback registration

The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_CRYP_RegisterCallback() or HAL_CRYP_RegisterXXXCallback() to register an interrupt callback.

Function @ref HAL_CRYP_RegisterCallback() allows to register following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_CRYP_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_CRYP_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit.

By default, after the @ref HAL_CRYP_Init() and when the state is HAL_CRYP_STATE_RESET all callbacks are set to the corresponding weak functions : examples @ref HAL_CRYP_InCpltCallback() , @ref HAL_CRYP_OutCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak function in the @ref HAL_CRYP_Init()/ @ref HAL_CRYP_DeInit() only when these callbacks are null (not registered beforehand). if not, MspInit or MspDeInit are not null, the @ref HAL_CRYP_Init() / @ref HAL_CRYP_DeInit() keep and use the user MspInit/MspDeInit functions (registered beforehand)

Callbacks can be registered/unregistered in HAL_CRYP_STATE_READY state only. Exception done MspInit/ MspDeInit callbacks that can be registered/unregistered in HAL_CRYP_STATE_READY or HAL_CRYP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_CRYP_RegisterCallback() before calling @ref HAL_CRYP_DeInit() or @ref HAL_CRYP_Init() function.

When The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Suspend/Resume feature

The compilation define USE_HAL_CRYP_SUSPEND_RESUME when set to 1 allows the user to resort to the suspend/resume feature. A low priority block processing can be suspended to process a high priority block instead. When the high priority block processing is over, the low priority block processing can be resumed, restarting from the point where it was suspended. This feature is applicable only in non-blocking interrupt mode.

User must resort to HAL_CRYP_Suspend() to suspend the low priority block processing. This API manages the hardware block processing suspension and saves all the internal data that will be needed to restart later on. Upon HAL_CRYP_Suspend() completion, the user can launch the processing of any other block (high priority block processing).

When the high priority block processing is over, user must invoke HAL_CRYP_Resume() to resume the low priority block processing. Ciphering (or deciphering) restarts from the suspension point and ends as usual.

HAL_CRYP_Suspend() reports an error when the suspension request is sent too late (i.e when the low priority block processing is about to end). There is no use to suspend the tag generation processing for authentication algorithms.

Note: If the key is written out of HAL scope (case pKey pointer set to NULL by the user), the block processing suspension/resumption mechanism is NOT applicable.

Note: If the Key and Initialization Vector are configured only once and configuration is skipped for consecutive processings (case KeyIVConfigSkip set to CRYP_KEYIVCONFIG_ONCE), the block processing suspension/ resumption mechanism is NOT applicable.

14.2.2 Initialization, de-initialization and Set and Get configuration functions

This section provides functions allowing to:

- Initialize the CRYP
- DeInitialize the CRYP
- Initialize the CRYP MSP
- DeInitialize the CRYP MSP
- configure CRYP (HAL_CRYP_SetConfig) with the specified parameters in the CRYP_ConfigTypeDef Parameters which are configured in This section are :
- Key size
- Data Type : 32,16, 8 or 1bit
- AlgoMode : - for CRYP1 peripheral : ECB and CBC in DES/TDES Standard ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard. - for TinyAES2 peripheral, only ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard are supported.
- Get CRYP configuration (HAL_CRYP_GetConfig) from the specified parameters in the CRYP_HandleTypeDef

This section contains the following APIs:

- ***HAL_CRYP_Init***
- ***HAL_CRYP_DeInit***
- ***HAL_CRYP_SetConfig***
- ***HAL_CRYP_GetConfig***
- ***HAL_CRYP_MspInit***
- ***HAL_CRYP_MspDeInit***

14.2.3 Encrypt Decrypt functions

This section provides API allowing to Encrypt/Decrypt Data following Standard DES/TDES or AES, and Algorithm configured by the user:

- Standard DES/TDES only supported by CRYP1 peripheral, below list of Algorithm supported : - Electronic Code Book(ECB) - Cipher Block Chaining (CBC)
- Standard AES supported by CRYP1 peripheral & TinyAES, list of Algorithm supported: - Electronic Code Book(ECB) - Cipher Block Chaining (CBC) - Counter mode (CTR) - Cipher Block Chaining (CBC) - Counter mode (CTR) - Galois/counter mode (GCM) - Counter with Cipher Block Chaining-Message(CCM)

Three processing functions are available:

- Polling mode : HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
- Interrupt mode : HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
- DMA mode : HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA

This section contains the following APIs:

- ***HAL_CRYP_Encrypt***
- ***HAL_CRYP_Decrypt***
- ***HAL_CRYP_Encrypt_IT***
- ***HAL_CRYP_Decrypt_IT***
- ***HAL_CRYP_Encrypt_DMA***
- ***HAL_CRYP_Decrypt_DMA***

14.2.4 CRYP IRQ handler management

This section provides CRYP IRQ handler and callback functions.

- HAL_CRYP_IRQHandler CRYP interrupt request
- HAL_CRYP_InCpltCallback input data transfer complete callback
- HAL_CRYP_OutCpltCallback output data transfer complete callback
- HAL_CRYP_ErrorCallback CRYP error callback
- HAL_CRYP_GetState return the CRYP state
- HAL_CRYP_GetError return the CRYP error code

This section contains the following APIs:

- ***HAL_CRYP_IRQHandler***
- ***HAL_CRYP_GetError***
- ***HAL_CRYP_GetState***
- ***HAL_CRYP_InCpltCallback***
- ***HAL_CRYP_OutCpltCallback***
- ***HAL_CRYP_ErrorCallback***

14.2.5 Detailed description of functions

HAL_CRYP_Init

Function name

HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP according to the specified parameters in the CRYP_ConfigTypeDef and creates the associated handle.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_DeInit

Function name

HAL_StatusTypeDef HAL_CRYP_DeInit (CRYP_HandleTypeDef * hcryp)

Function description

De-Initializes the CRYP peripheral.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_MspInit

Function name

void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_MspDeInit

Function name

void HAL_CRYP_MspDeInit (CRYP_HandleTypeDef * hcryp)

Function description

DeInitializes CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_SetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_SetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Configure the CRYP according to the specified parameters in the CRYP_ConfigTypeDef.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure
- **pConf:** pointer to a CRYP_ConfigTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_GetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_GetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Get CRYP Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a CRYP_ConfigTypeDef structure
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_Encrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output, uint32_t Timeout)

Function description

Encryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(ciphertext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Decrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output, uint32_t Timeout)

Function description

Decryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(plaintext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Encrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Encryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Decryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_Encrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Encryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Decryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_IRQHandler

Function name

void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)

Function description

This function handles cryptographic interrupt request.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_GetState

Function name

HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)

Function description

Returns the CRYP state.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **HAL:** state

HAL_CRYP_InCpltCallback

Function name

void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Input FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_OutCpltCallback

Function name

void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Output FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_ErrorCallback

Function name

void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)

Function description

CRYP error callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_GetError

Function name

uint32_t HAL_CRYP_GetError (CRYP_HandleTypeDef * hcryp)

Function description

Return the CRYP error code.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for the CRYP peripheral

Return values

- **CRYP:** error code

14.3 CRYP Firmware driver defines

The following section lists the various define and macros of the module.

14.3.1 CRYP

CRYP

CRYP Algorithm Mode

CRYP_AES_ECB

Electronic codebook chaining algorithm

CRYP_AES_CBC

Cipher block chaining algorithm

CRYP_AES_CTR

Counter mode chaining algorithm

CRYP_AES_GCM_GMAC

Galois counter mode - Galois message authentication code

CRYP_AES_CCM

Counter with Cipher Mode

CRYP Key and IV Configuration Skip Mode

CRYP_KEYIVCONFIG_ALWAYS

Peripheral Key and IV configuration to do systematically

CRYP_KEYIVCONFIG_ONCE

Peripheral Key and IV configuration to do only once

CRYP Data Type

CRYP_DATATYPE_32B

32-bit data type (no swapping)

CRYP_DATATYPE_16B

16-bit data type (half-word swapping)

CRYP_DATATYPE_8B

8-bit data type (byte swapping)

CRYP_DATATYPE_1B

1-bit data type (bit swapping)

CRYP Data Width Unit

CRYP_DATAWIDTHUNIT_WORD

By default, size unit is word

CRYP_DATAWIDTHUNIT_BYTE

By default, size unit is byte

CRYP Error Definition

HAL_CRYP_ERROR_NONE

No error

HAL_CRYP_ERROR_WRITE

Write error

HAL_CRYP_ERROR_READ

Read error

HAL_CRYP_ERROR_DMA

DMA error

HAL_CRYP_ERROR_BUSY

Busy flag error

HAL_CRYP_ERROR_TIMEOUT

Timeout error

HAL_CRYP_ERROR_NOT_SUPPORTED

Not supported mode

HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE

Sequence are not respected only for GCM or CCM

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE

Description:

- Reset CRYP handle state.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_ENABLE

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_DISABLE

CRYP_FLAG_MASK

Description:

- Check whether the specified CRYP status flag is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values for TinyAES:
  - CRYP_FLAG_BUSY GCM process suspension forbidden
  - CRYP_IT_WRERR Write Error
  - CRYP_IT_RDERR Read Error
  - CRYP_IT_CCF Computation Complete This parameter can be one of the following values for CRYP:
  - CRYP_FLAG_BUSY: The CRYP core is currently processing a block of data or a key preparation (for AES decryption).
  - CRYP_FLAG_IFEM: Input FIFO is empty
  - CRYP_FLAG_IFNF: Input FIFO is not full
  - CRYP_FLAG_INRIS: Input FIFO service raw interrupt is pending
  - CRYP_FLAG_OFNE: Output FIFO is not empty
  - CRYP_FLAG_OFFU: Output FIFO is full
  - CRYP_FLAG_OUTRIS: Input FIFO service raw interrupt is pending

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_GET_FLAG

__HAL_CRYP_CLEAR_FLAG

Description:

- Clear the CRYP pending status flag.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - CRYP_ERR_CLEAR Read (RDERR) or Write Error (WRERR) Flag Clear
  - CRYP_CCF_CLEAR Computation Complete Flag (CCF) Clear

Return value:

- None

__HAL_CRYP_GET_IT_SOURCE

Description:

- Check whether the specified CRYP interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP interrupt source to check This parameter can be one of the following values for TinyAES:
  - CRYP_IT_ERRIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt

Return value:

- State: of interruption (TRUE or FALSE).

__HAL_CRYP_GET_IT

Description:

- Check whether the specified CRYP interrupt is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: specifies the interrupt to check. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_WRERR Write Error
  - CRYP_IT_RDERR Read Error
  - CRYP_IT_CCF Computation Complete This parameter can be one of the following values for CRYP:
  - CRYP_IT_INI: Input FIFO service masked interrupt status
  - CRYP_IT_OUTI: Output FIFO service masked interrupt status

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_ERRIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt This parameter can be one of the following values for CRYP: @ CRYP_IT_INI : Input FIFO service interrupt mask. @ CRYP_IT_OUTI : Output FIFO service interrupt mask.CRYP interrupt.

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_ERRIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt This parameter can be one of the following values for CRYP: @ CRYP_IT_INI : Input FIFO service interrupt mask. @ CRYP_IT_OUTI : Output FIFO service interrupt mask.CRYP interrupt.

Return value:

- None

CRYP Flags

CRYP_FLAG_BUSY

GCM process suspension forbidden

CRYP_FLAG_WRERR

Write Error

CRYP_FLAG_RDERR

Read error

CRYP_FLAG_CCF

Computation completed

CRYP_CCF_CLEAR

Computation Complete Flag Clear

CRYP_ERR_CLEAR

Error Flag Clear

CRYP Header Width Unit

CRYP_HEADERWIDTHUNIT_WORD

By default, header size unit is word

CRYP_HEADERWIDTHUNIT_BYTE

By default, header size unit is byte

CRYP Interrupt

CRYP_IT_CCFIE

Computation Complete interrupt enable

CRYP_IT_ERRIE

Error interrupt enable

CRYP_IT_WRERR

Write Error

CRYP_IT_RDERR

Read Error

CRYP_IT_CCF

Computation completed

CRYP Private macros to check input parameters

IS_CRYP_ALGORITHM

IS_CRYP_KEYSIZE

IS_CRYP_DATATYPE

IS_CRYP_INIT

IS_CRYP_BUFFERSIZE

CRYP Key Size

CRYP_KEYSIZE_128B

128-bit long key

CRYP_KEYSIZE_256B

256-bit long key

15 HAL CRYP Extension Driver

15.1 CRYPEx Firmware driver API description

The following section lists the various functions of the CRYPEx library.

15.1.1 Extended AES processing functions

This section provides functions allowing to generate the authentication TAG in Polling mode

1. HAL_CRYPEx_AESGCM_GenerateAuthTAG
2. HAL_CRYPEx_AESCCM_GenerateAuthTAG they should be used after Encrypt/Decrypt operation.

This section contains the following APIs:

- ***HAL_CRYPEx_AESGCM_GenerateAuthTAG***
- ***HAL_CRYPEx_AESCCM_GenerateAuthTAG***

15.1.2 Key Derivation functions

This section provides functions allowing to Enable or Disable the the AutoKeyDerivation parameter in CRYP_HandleTypeDef structure These function are allowed only in TinyAES peripheral.

This section contains the following APIs:

- ***HAL_CRYPEx_EnableAutoKeyDerivation***
- ***HAL_CRYPEx_DisableAutoKeyDerivation***

15.1.3 Detailed description of functions

HAL_CRYPEx_AESGCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESGCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)

Function description

generate the GCM authentication TAG.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **AuthTag:** Pointer to the authentication buffer
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_AESCCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)

Function description

AES CCM Authentication TAG generation.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **AuthTag:** Pointer to the authentication buffer
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_EnableAutoKeyDerivation

Function name

void HAL_CRYPEx_EnableAutoKeyDerivation (CRYP_HandleTypeDef * hcryp)

Function description

AES enable key derivation functions.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure.

HAL_CRYPEx_DisableAutoKeyDerivation

Function name

void HAL_CRYPEx_DisableAutoKeyDerivation (CRYP_HandleTypeDef * hcryp)

Function description

AES disable key derivation functions.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure.

16 HAL DAC Generic Driver

16.1 DAC Firmware driver registers structures

16.1.1 DAC_HandleTypeDef

DAC_HandleTypeDef is defined in the stm32g4xx_hal_dac.h

Data Fields

- ***DAC_TypeDef * Instance***
- ***__IO HAL_DAC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * DMA_Handle1***
- ***DMA_HandleTypeDef * DMA_Handle2***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DAC_TypeDef* DAC_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_DAC_StateTypeDef DAC_HandleTypeDef::State***
  DAC communication state
- ***HAL_LockTypeDef DAC_HandleTypeDef::Lock***
  DAC locking object
- ***DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle1***
  Pointer DMA handler for channel 1
- ***DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle2***
  Pointer DMA handler for channel 2
- ***__IO uint32_t DAC_HandleTypeDef::ErrorCode***
  DAC Error code

16.1.2 DAC_SampleAndHoldConfTypeDef

DAC_SampleAndHoldConfTypeDef is defined in the stm32g4xx_hal_dac.h

Data Fields

- ***uint32_t DAC_SampleTime***
- ***uint32_t DAC_HoldTime***
- ***uint32_t DAC_RefreshTime***

Field Documentation

- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_SampleTime***
  Specifies the Sample time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_HoldTime***
  Specifies the hold time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_RefreshTime***
  Specifies the refresh time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 255

16.1.3 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the stm32g4xx_hal_dac.h

Data Fields

- ***uint32_t DAC_HighFrequency***
- ***FunctionalState DAC_DMADoubleDataMode***
- ***FunctionalState DAC_SignedFormat***
- ***uint32_t DAC_SampleAndHold***
- ***uint32_t DAC_Trigger***
- ***uint32_t DAC_Trigger2***
- ***uint32_t DAC_OutputBuffer***
- ***uint32_t DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_UserTrimming***
- ***uint32_t DAC_TrimmingValue***
- ***DAC_SampleAndHoldConfTypeDef DAC_SampleAndHoldConfig***

Field Documentation

- ***uint32_t DAC_ChannelConfTypeDef::DAC_HighFrequency***

  Specifies the frequency interface mode This parameter can be a value of ***DAC_HighFrequency***
- ***FunctionalState DAC_ChannelConfTypeDef::DAC_DMADoubleDataMode***

  Specifies if DMA double data mode should be enabled or not for the selected channel. This parameter can be ENABLE or DISABLE
- ***FunctionalState DAC_ChannelConfTypeDef::DAC_SignedFormat***

  Specifies if signed format should be used or not for the selected channel. This parameter can be ENABLE or DISABLE
- ***uint32_t DAC_ChannelConfTypeDef::DAC_SampleAndHold***

  Specifies whether the DAC mode. This parameter can be a value of ***DAC_SampleAndHold***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger***

  Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***. Note: In case of sawtooth wave generation, this trigger corresponds to the reset trigger.
- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger2***

  Specifies the external secondary trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***. Note: In case of sawtooth wave generation, this trigger corresponds to the step trigger.
- ***uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer***

  Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of ***DAC_output_buffer***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_ConnectOnChipPeripheral***

  Specifies whether the DAC output is connected or not to on chip peripheral . This parameter can be a value of ***DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_UserTrimming***

  Specifies the trimming mode This parameter must be a value of ***DAC_UserTrimming*** DAC_UserTrimming is either factory or user trimming
- ***uint32_t DAC_ChannelConfTypeDef::DAC_TrimmingValue***

  Specifies the offset trimming value i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***DAC_SampleAndHoldConfTypeDef DAC_ChannelConfTypeDef::DAC_SampleAndHoldConfig***

  Sample and Hold settings

16.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

16.2.1 DAC Peripheral features

DAC Channels

STM32G4 devices integrate up to seven 12-bit Digital Analog Converters, up to six of them grouped by pair forming a DAC instance. The 2 converters of an single instance (i.e. channel1 & channel2) can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 as output (not for all) or connected to on-chip peripherals (ex. comparators, operational amplifier).
2. DAC channel2 with DAC_OUT2 as output (not for all) or connected to on-chip peripherals (ex. comparators, operational amplifier). Note: when an instance only includes one converter, only independent mode is supported by this converter. STM32G4 instances & converters availability and output PIO mapping (DAC_OUTx): ---------- | DAC1 | DAC2 | DAC3 | DAC4 | ---------- Channel 1 | | YES | YES | YES | YES | DAC_OUT1 | PA4 | PA6 | - | - ---------- Channel 2 | | YES | NO | YES | YES | DAC_OUT2 | PA5 | - | - | - ---------- Note: On this STM32 serie, all devices do not include each DAC instances listed above. Refer to device datasheet for DACx instance availability.

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM1, TIM2, TIM3, TIM4, TIM6, TIM7, TIM8 and TIM15 (DAC_TRIGGER_T2_TRGO, DAC_TRIGGER_T3_TRGO...)
3. Software using DAC_TRIGGER_SOFTWARE
4. HRTimer TRGO: HRTIM1 (1) (DAC_TRIGGER_HRTIM_TRG01, DAC_TRIGGER_HRTIM_TRG02...)

Specific triggers for sawtooth generation:

1. External event: EXTI Line 10 (any GPIOx_PIN_10) using DAC_TRIGGER_EXT_IT10. The used pin (GPIOx_PIN_10) must be configured in input mode.
2. HRTimer Step & Reset: HRTIM1 (1) (DAC_TRIGGER_HRTIM_RST_TRG1, DAC_TRIGGER_HRTIM_STEP_TRG1...) Note: On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Note: Refer to the device datasheet for more details about output impedance value with and without output buffer.

DAC connect feature

Each DAC channel can be connected internally. To connect, use sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_INTERNAL; or sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_BOTH;

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel2 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC Sample and Hold feature

For each converter, 2 modes are supported: normal mode and "sample and hold" mode (i.e. low power mode). In the sample and hold mode, the DAC core converts data, then holds the converted voltage on a capacitor. When not converting, the DAC cores and buffer are completely turned off between samples and the DAC output is tri-stated, therefore reducing the overall power consumption. A new stabilization period is needed before each new conversion. The sample and hold allow setting internal or external voltage @ low power consumption cost (output value can be at any given rate either by CPU or DMA). The Sample and hold block and registers uses either LSI & run in several power modes: run mode, sleep mode, low power run, low power sleep mode & stop1 mode. Low power stop1 mode allows only static conversion. To enable Sample and Hold mode Enable LSI using HAL_RCC_OscConfig with RCC_OSCILLATORTYPE_LSI & RCC_LSI_ON parameters. Use DAC_InitStructure.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_ENABLE; & DAC_ChannelConfTypeDef.DAC_SampleAndHoldConfig.DAC_SampleTime, DAC_HoldTime & DAC_RefreshTime;

DAC calibration feature

1. The 2 converters (channel1 & channel2) provide calibration capabilities.
   - Calibration aims at correcting some offset of output buffer.
   - The DAC uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_DACEx_SelfCalibrate.
   - HAL_DACEx_SelfCalibrate:
     - Runs automatically the calibration.
     - Enables the user trimming mode
     - Updates a structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave
3. Sawtooth wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:

DAC_OUTx = VREF+ * DOR / 4095

- with DOR is the Data Output Register

VEF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMAMUX request can be generated when an external trigger (but not a software trigger) occurs if DMAMUX requests are enabled using HAL_DAC_Start_DMA(). DMAMUX requests are mapped as following:
---------- | DAC1 | DAC2 | DAC3 | DAC4 |
---------- Channel 1 | | 6 | 41 | 102 | 104
---------- Channel 2 | | 7 | - | 103 | 105
---------- Note: On this STM32 serie, all devices do not include each DAC instances listed above. Refer to device datasheet for DACx instance availability.

High frequency interface mode

The high frequency interface informs DAC instance about the bus frequency in use. It is mandatory information for DAC (as internal timing of DAC is bus frequency dependent) provided thanks to parameter DAC_HighFrequency handled in HAL_DAC_ConfigChannel () function. Use of DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC value of DAC_HighFrequency is recommended function figured out the correct setting. The high frequency mode is same for all converters of a same DAC instance. Either same parameter DAC_HighFrequency is used for all DAC converters or again self DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC detection parameter.

Note: For Dual mode and specific signal (Sawtooth, triangle and noise) generation please refer to Extended Features Driver description

16.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- If available & needed, configure DAC_OUTx (DAC_OUT1, DAC_OUT2) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Calibration mode IO operation

- Retrieve the factory trimming (calibration settings) using HAL_DACEx_GetTrimOffset()
- Run the calibration using HAL_DACEx_SelfCalibrate()
- Update the trimming while DAC running using HAL_DACEx_SetUserTrimming()

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion First issued trigger will start the conversion of the value previously set by HAL_DAC_SetValue().
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1

- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1()
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

Callback registration

The compilation define USE_HAL_DAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_DAC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function @ref HAL_DAC_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit.
- All Callbacks This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_DAC_Init and if the state is HAL_DAC_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_DAC_Init and @ref HAL_DAC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_DAC_Init and @ref HAL_DAC_DeInit keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_DAC_RegisterCallback before calling @ref HAL_DAC_DeInit or @ref HAL_DAC_Init function. When The compilation define USE_HAL_DAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

DAC HAL driver macros list

Below the list of most used macros in DAC HAL driver.

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

Note: You can refer to the DAC HAL driver header file for more useful macros

16.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:

- ***HAL_DAC_Init***
- ***HAL_DAC_DeInit***
- ***HAL_DAC_MspInit***
- ***HAL_DAC_MspDeInit***

16.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.

This section contains the following APIs:

- ***HAL_DAC_Start***
- ***HAL_DAC_Stop***
- ***HAL_DAC_Start_DMA***
- ***HAL_DAC_Stop_DMA***
- ***HAL_DAC_IRQHandler***
- ***HAL_DAC_SetValue***
- ***HAL_DAC_ConvCpltCallbackCh1***
- ***HAL_DAC_ConvHalfCpltCallbackCh1***
- ***HAL_DAC_ErrorCallbackCh1***
- ***HAL_DAC_DMAUnderrunCallbackCh1***

16.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- ***HAL_DAC_GetValue***
- ***HAL_DAC_ConfigChannel***

16.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- ***HAL_DAC_GetState***
- ***HAL_DAC_GetError***

16.2.7 Detailed description of functions

HAL_DAC_Init

Function name

HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC peripheral according to the specified parameters in the DAC_InitStruct and initialize the associated handle.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_DeInit

Function name

HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)

Function description

Deinitialize the DAC peripheral registers to their default reset values.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_MspInit

Function name

void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_MspDeInit

Function name

void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)

Function description

DeInitialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_Start

Function name

HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **HAL:** status

HAL_DAC_Stop

Function name

HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **HAL:** status

HAL_DAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DAC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **HAL:** status

HAL_DAC_IRQHandler

Function name

void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)

Function description

Handles DAC interrupt request This function uses the interruption of DMA underrun.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_SetValue

Function name

HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Alignment:** Specifies the data alignment. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values

- **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name

void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name

void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ErrorCallbackCh1

Function name

void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_DMAUnderrunCallbackCh1

Function name

void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_GetValue

Function name

uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Returns the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **The:** selected DAC channel data output value.

HAL_DAC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Configures the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **HAL:** status

Notes

- By calling this function, the high frequency interface mode (HFSEL bits) will be set. This parameter scope is the DAC instance. As the function is called for each channel, the DAC high frequency interface mode of sConfig must be the same at each call. (or DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC self detect).

HAL_DAC_GetState

Function name

HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)

Function description

return the DAC handle state

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** state

HAL_DAC_GetError

Function name

uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)

Function description

Return the DAC error code.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **DAC:** Error Code

DAC_DMAConvCpltCh1

Function name

void DAC_DMAConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh1

Function name

void DAC_DMAErrorCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh1

Function name

void DAC_DMAHalfConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

16.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

16.3.1 DAC

DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC ConnectOnChipPeripheral

DAC_CHIPCONNECT_EXTERNAL

DAC_CHIPCONNECT_INTERNAL

DAC_CHIPCONNECT_BOTH

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE

No error

HAL_DAC_ERROR_DMAUNDERRUNCH1

DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERRUNCH2

DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA

DMA error

HAL_DAC_ERROR_TIMEOUT

Timeout error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC channel.

Return value:

- None

DAC_DHR12R1_ALIGNMENT

Description:

- Set DHR12R1 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12R2_ALIGNMENT

Description:

- Set DHR12R2 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12RD_ALIGNMENT

Description:

- Set DHR12RD alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

__HAL_DAC_ENABLE_IT

Description:

- Enable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt (1)

Return value:

- None

__HAL_DAC_DISABLE_IT

Description:

- Disable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt (1)

Return value:

- None

__HAL_DAC_GET_IT_SOURCE

Description:

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1: DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2: DAC channel 2 DMA underrun interrupt (1)

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG

Description:

- Get the selected DAC's flag status.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to get. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2: DAC channel 2 DMA underrun flag (1)
  - DAC_FLAG_DAC1RDY: DAC channel 1 ready status flag
  - DAC_FLAG_DAC2RDY: DAC channel 2 ready status flag (1)

Return value:

- None

__HAL_DAC_CLEAR_FLAG

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to clear. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2: DAC channel 2 DMA underrun flag (1)

Return value:

- None

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

DAC_FLAG_DAC1RDY

DAC_FLAG_DAC2RDY

DAC high frequency interface mode

DAC_HIGH_FREQUENCY_INTERFACE_MODE_DISABLE

High frequency interface mode disabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_80MHZ

High frequency interface mode compatible to AHB>80MHz enabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_160MHZ

High frequency interface mode compatible to AHB>160MHz enabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC

High frequency interface mode automatic

DAC IT definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC power mode

DAC_SAMPLEANDHOLD_DISABLE

DAC_SAMPLEANDHOLD_ENABLE

DAC trigger selection

DAC_TRIGGER_NONE

DAC (all) conversion is automatic once the DAC_DHRxxxx register has been loaded, and not by external trigger

DAC_TRIGGER_SOFTWARE

DAC (all) conversion started by software trigger for DAC channel

DAC_TRIGGER_T1_TRGO

DAC3: TIM1 TRGO selected as external conversion trigger for DAC channel.

DAC_TRIGGER_T8_TRGO

DAC1/2/4: TIM8 TRGO selected as external conversion trigger for DAC channel. Refer to device datasheet for DACx availability.

DAC_TRIGGER_T7_TRGO

DAC (all): TIM7 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T15_TRGO

DAC (all): TIM15 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T2_TRGO

DAC (all): TIM2 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T4_TRGO

DAC (all): TIM4 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_EXT_IT9

DAC (all): EXTI Line9 event selected as external conversion trigger for DAC channel. Note: only to be used as update or reset (sawtooth generation) trigger

DAC_TRIGGER_EXT_IT10

DAC (all): EXTI Line10 event selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger

DAC_TRIGGER_T6_TRGO

DAC (all): TIM6 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T3_TRGO

DAC (all): TIM3 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_HRTIM_RST_TRG1

DAC (all): HRTIM RST TRIG 1 selected as external conversion trigger for DAC channel. Note: only to be used as reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_STEP_TRG1

DAC (all): HRTIM STEP TRIG 1 selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_RST_TRG2

DAC (all): HRTIM RST TRIG 2 selected as external conversion trigger for DAC channel. Note: only to be used as reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_STEP_TRG2

DAC (all): HRTIM STEP TRIG 2 selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_RST_TRG3

DAC (all): HRTIM RST TRIG 3 selected as external conversion trigger for DAC channel. Note: only to be used as reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_STEP_TRG3

DAC (all): HRTIM STEP TRIG 3 selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_RST_TRG4

DAC (all): HRTIM RST TRIG 4 selected as external conversion trigger for DAC channel. Note: only to be used as reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_STEP_TRG4

DAC (all): HRTIM STEP TRIG 4 selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_RST_TRG5

DAC (all): HRTIM RST TRIG 5 selected as external conversion trigger for DAC channel. Note: only to be used as reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_STEP_TRG5

DAC (all): HRTIM STEP TRIG 5 selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_RST_TRG6

DAC (all): HRTIM RST TRIG 6 selected as external conversion trigger for DAC channel. Note: only to be used as reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_STEP_TRG6

DAC (all): HRTIM STEP TRIG 6 selected as external conversion trigger for DAC channel. Note: only to be used as step (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_TRG01

DAC1&4: HRTIM TRIG OUT 1 selected as external conversion trigger for DAC channel. Note: only to be used as update or reset (sawtooth generation) trigger. Refer to device datasheet for DACx instance availability. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC_TRIGGER_HRTIM_TRG02

DAC2: HRTIM TRIG OUT 1 selected as external conversion trigger for DAC channel. Note: only to be used as update or reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported and DAC2 instance present (refer to device datasheet for supported features list and DAC2 instance availability)

DAC_TRIGGER_HRTIM_TRG03

DAC3: HRTIM TRIG OUT 1 selected as external conversion trigger for DAC channel. Note: only to be used as update or reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC User Trimming

DAC_TRIMMING_FACTORY

Factory trimming

DAC_TRIMMING_USER

User trimming

17 HAL DAC Extension Driver

17.1 DACEx Firmware driver API description

The following section lists the various functions of the DACEx library.

17.1.1 How to use this driver

Dual mode IO operation

- Use HAL_DACEx_DualStart() to enable both channels and start the conversion in dual mode operation.
  If the software trigger is selected, using HAL_DACEx_DualStart() starts the conversion of the value previously set by HAL_DACEx_DualSetValue().
- Use HAL_DACEx_DualStop() to disable both channels and stop the conversion in dual mode operation.
- Use HAL_DACEx_DualStart_DMA() to enable both channels and start the conversion in dual mode operation using the DMA.to feed the DAC converters.
  The first trigger issued starts the conversion of the value previously set by HAL_DACEx_DualSetValue().
  The same callbacks that are used in signel mode are called in dual mode to notify transfer completion (half complte or complete), errors or underrun.
- Use HAL_DACEx_DualStop_DMA() to disable both channels and stop the conversion in dual mode operation using the DMA.to feed the DAC converters.
- When Dual mode is enabled (that is DAC Channel 1 and Channel 2 are used simultaneously), use HAL_DACEx_DualGetValue() to get digital data to be converted and HAL_DACEx_DualSetValue() to set digital value to converted simultaneously on Channel 1 and Channel 2.

Signal generation operation

- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.
- Use HAL_DACEx_SawtoothWaveGenerate() to generate sawtooth signal.
- Use HAL_DACEx_SawtoothWaveDataReset() to reset sawtooth wave.
- Use HAL_DACEx_SawtoothWaveDataStep() to step sawtooth wave.
- Use HAL_DACEx_SetCalibrate to calibrate one DAC channel.
- Use HAL_DACEx_SetUserTrimming to set user trimming value.
- Use HAL_DACEx_GetTrimOffset to retrieve trimming value (factory setting after reset, user setting if HAL_DACEx_SetUserTrimming has been used at least once after reset).

17.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- ***HAL_DACEx_DualStart***
- ***HAL_DACEx_DualStop***
- ***HAL_DACEx_DualStart_DMA***
- ***HAL_DACEx_DualStop_DMA***
- ***HAL_DACEx_TriangleWaveGenerate***

- *HAL_DACEx_NoiseWaveGenerate*
- *HAL_DACEx_SawtoothWaveGenerate*
- *HAL_DACEx_SawtoothWaveDataReset*
- *HAL_DACEx_SawtoothWaveDataStep*
- *HAL_DACEx_DualSetValue*
- *HAL_DACEx_ConvCpltCallbackCh2*
- *HAL_DACEx_ConvHalfCpltCallbackCh2*
- *HAL_DACEx_ErrorCallbackCh2*
- *HAL_DACEx_DMAUnderrunCallbackCh2*
- *HAL_DACEx_SelfCalibrate*
- *HAL_DACEx_SetUserTrimming*
- *HAL_DACEx_GetTrimOffset*
- *HAL_DACEx_DualGetValue*

17.1.3 Peripheral Control functions

This section provides functions allowing to:

- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DACEx_DualGetValue*
- *HAL_DACEx_SelfCalibrate*
- *HAL_DACEx_SetUserTrimming*
- *HAL_DACEx_GetTrimOffset*

17.1.4 Detailed description of functions

HAL_DACEx_TriangleWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
  - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
  - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
  - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
  - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
  - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
  - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
  - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
  - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
  - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
  - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
  - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
  - DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
  - DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
  - DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values

- **HAL:** status

HAL_DACEx_SawtoothWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_SawtoothWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Polarity, uint32_t ResetData, uint32_t StepData)

Function description

Enable or disable the selected DAC channel sawtooth wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Polarity:** polarity to be used for wave generation. This parameter can be one of the following values:
  - DAC_SAWTOOTH_POLARITY_DECREMENT
  - DAC_SAWTOOTH_POLARITY_INCREMENT
- **ResetData:** Sawtooth wave reset value. Range is from 0 to DAC full range 4095 (0xFFF)
- **StepData:** Sawtooth wave step value. 12.4 bit format, unsigned: 12 bits exponent / 4 bits mantissa Step value step is 1/16 = 0.0625 Step value range is 0.0000 to 4095.9375 (0xFFF.F)

Return values

- **HAL:** status

Notes

- Sawtooth reset and step triggers are configured by calling HAL_DAC_ConfigChannel

HAL_DACEx_SawtoothWaveDataReset

Function name

HAL_StatusTypeDef HAL_DACEx_SawtoothWaveDataReset (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Trig sawtooth wave reset.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **HAL:** status

Notes

- This function allows to reset sawtooth wave in case of SW trigger has been configured for this usage.

HAL_DACEx_SawtoothWaveDataStep

Function name

HAL_StatusTypeDef HAL_DACEx_SawtoothWaveDataStep (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Trig sawtooth wave step.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **HAL:** status

Notes

- This function allows to generate step in sawtooth wave in case of SW trigger has been configured for this usage.

HAL_DACEx_DualStart

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart (DAC_HandleTypeDef * hdac)

Function description

Enables DAC and starts conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStop

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop (DAC_HandleTypeDef * hdac)

Function description

Disables DAC and stop conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStart_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of both channel 1 and 2 of the same DAC.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that will request data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DACEx_DualStop_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion both channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that requests data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DACEx_DualSetValue

Function name

HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)

Function description

Set the specified data holding register value for dual DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel1 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel2 to be loaded in the selected data holding register.

Return values

- **HAL:** status

Notes

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_DualGetValue

Function name

uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)

Function description

Return the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **The:** selected DAC channel data output value.

HAL_DACEx_ConvCpltCallbackCh2

Function name

void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name

void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ErrorCallbackCh2

Function name

void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name

void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_DACEx_SelfCalibrate (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Run the self calibration of one DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC channel configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Updates:** DAC_TrimmingValue. , DAC_UserTrimming set to DAC_UserTrimming
- **HAL:** status

Notes

- Calibration runs about 7 ms.

HAL_DACEx_SetUserTrimming

Function name

HAL_StatusTypeDef HAL_DACEx_SetUserTrimming (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel, uint32_t NewTrimmingValue)

Function description

Set the trimming mode and trimming value (user trimming mode applied).

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure updated with new DAC trimming value.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **NewTrimmingValue:** DAC new trimming value

Return values

- **HAL:** status

HAL_DACEx_GetTrimOffset

Function name

uint32_t HAL_DACEx_GetTrimOffset (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Return the DAC trimming value.

Parameters

- **hdac:** DAC handle
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Trimming:** value : range: 0->31

DAC_DMAConvCpltCh2

Function name

void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh2

Function name

void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh2

Function name

void DAC_DMAHalfConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

17.2 DACEx Firmware driver defines

The following section lists the various define and macros of the module.

17.2.1 DACEx

DACEx

DACEx lfsrunmask triangle amplitude

DAC_LFSRUNMASK_BIT0

Unmask DAC channel LFSR bit0 for noise wave generation

DAC_LFSRUNMASK_BITS1_0

Unmask DAC channel LFSR bit[1:0] for noise wave generation

DAC_LFSRUNMASK_BITS2_0

Unmask DAC channel LFSR bit[2:0] for noise wave generation

DAC_LFSRUNMASK_BITS3_0

Unmask DAC channel LFSR bit[3:0] for noise wave generation

DAC_LFSRUNMASK_BITS4_0

Unmask DAC channel LFSR bit[4:0] for noise wave generation

DAC_LFSRUNMASK_BITS5_0

Unmask DAC channel LFSR bit[5:0] for noise wave generation

DAC_LFSRUNMASK_BITS6_0

Unmask DAC channel LFSR bit[6:0] for noise wave generation

DAC_LFSRUNMASK_BITS7_0

Unmask DAC channel LFSR bit[7:0] for noise wave generation

DAC_LFSRUNMASK_BITS8_0

Unmask DAC channel LFSR bit[8:0] for noise wave generation

DAC_LFSRUNMASK_BITS9_0

Unmask DAC channel LFSR bit[9:0] for noise wave generation

DAC_LFSRUNMASK_BITS10_0

Unmask DAC channel LFSR bit[10:0] for noise wave generation

DAC_LFSRUNMASK_BITS11_0

Unmask DAC channel LFSR bit[11:0] for noise wave generation

DAC_TRIANGLEAMPLITUDE_1

Select max triangle amplitude of 1

DAC_TRIANGLEAMPLITUDE_3

Select max triangle amplitude of 3

DAC_TRIANGLEAMPLITUDE_7

Select max triangle amplitude of 7

DAC_TRIANGLEAMPLITUDE_15

Select max triangle amplitude of 15

DAC_TRIANGLEAMPLITUDE_31

Select max triangle amplitude of 31

DAC_TRIANGLEAMPLITUDE_63

Select max triangle amplitude of 63

DAC_TRIANGLEAMPLITUDE_127

Select max triangle amplitude of 127

DAC_TRIANGLEAMPLITUDE_255

Select max triangle amplitude of 255

DAC_TRIANGLEAMPLITUDE_511

Select max triangle amplitude of 511

DAC_TRIANGLEAMPLITUDE_1023

Select max triangle amplitude of 1023

DAC_TRIANGLEAMPLITUDE_2047

Select max triangle amplitude of 2047

DAC_TRIANGLEAMPLITUDE_4095

Select max triangle amplitude of 4095

DAC Sawtooth polarity mode

DAC_SAWTOOTH_POLARITY_DECREMENT

Sawtooth wave generation, polarity is decrement

DAC_SAWTOOTH_POLARITY_INCREMENT

Sawtooth wave generation, polarity is increment

18 HAL DMA Generic Driver

18.1 DMA Firmware driver registers structures

18.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32g4xx_hal_dma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t Direction***
- ***uint32_t PeriphInc***
- ***uint32_t MemInc***
- ***uint32_t PeriphDataAlignment***
- ***uint32_t MemDataAlignment***
- ***uint32_t Mode***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***

  Specifies the request selected for the specified channel. This parameter can be a value of ***DMA_request***
- ***uint32_t DMA_InitTypeDef::Direction***

  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc***

  Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of ***DMA_Peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::MemInc***

  Specifies whether the memory address register should be incremented or not. This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***

  Specifies the Peripheral data width. This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***

  Specifies the Memory data width. This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode***

  Specifies the operation mode of the DMAy Channelx. This parameter can be a value of ***DMA_mode***

  Note:

  – The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- ***uint32_t DMA_InitTypeDef::Priority***

  Specifies the software priority for the DMAy Channelx. This parameter can be a value of ***DMA_Priority_level***

18.1.2 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32g4xx_hal_dma.h

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DMA_StateTypeDef State***
- ***void * Parent***

- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***__IO uint32_t ErrorCode***
- ***DMA_TypeDef * DmaBaseAddress***
- ***uint32_t ChannelIndex***
- ***DMAMUX_Channel_TypeDef * DMAmuxChannel***
- ***DMAMUX_ChannelStatus_TypeDef * DMAmuxChannelStatus***
- ***uint32_t DMAmuxChannelStatusMask***
- ***DMAMUX_RequestGen_TypeDef * DMAmuxRequestGen***
- ***DMAMUX_RequestGenStatus_TypeDef * DMAmuxRequestGenStatus***
- ***uint32_t DMAmuxRequestGenStatusMask***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA Half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer abort callback
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA Error code
- ***DMA_TypeDef* __DMA_HandleTypeDef::DmaBaseAddress***
  DMA Channel Base Address
- ***uint32_t __DMA_HandleTypeDef::ChannelIndex***
  DMA Channel Index
- ***DMAMUX_Channel_TypeDef* __DMA_HandleTypeDef::DMAmuxChannel***
  Register base address
- ***DMAMUX_ChannelStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxChannelStatus***
  DMAMUX Channels Status Base Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxChannelStatusMask***
  DMAMUX Channel Status Mask
- ***DMAMUX_RequestGen_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGen***
  DMAMUX request generator Base Address
- ***DMAMUX_RequestGenStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGenStatus***
  DMAMUX request generator Address

- ***uint32_t __DMA_HandleTypeDef::DMAmuxRequestGenStatusMask***
  DMAMUX request generator Status mask

18.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

18.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM / FLASH memories: no initialization is necessary). Please refer to the Reference manual for connection between peripherals and DMA requests.
2. For a given Channel, program the required configuration through the following parameters: Channel request, Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode using HAL_DMA_Init() function. Prior to HAL_DMA_Init the peripheral clock shall be enabled for both DMA & DMAMUX thanks to:
   a. DMA1 or DMA2: __HAL_RCC_DMA1_CLK_ENABLE() or __HAL_RCC_DMA2_CLK_ENABLE() ;
   b. DMAMUX1: __HAL_RCC_DMAMUX1_CLK_ENABLE();
3. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
4. Use HAL_DMA_Abort() function to abort the current transfer

Note: In Memory-to-Memory transfer mode, Circular mode is not allowed.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function to register callbacks with HAL_DMA_RegisterCallback().

DMA HAL driver macros list

Below the list of macros in DMA HAL driver.

- __HAL_DMA_ENABLE: Enable the specified DMA Channel.
- __HAL_DMA_DISABLE: Disable the specified DMA Channel.
- __HAL_DMA_GET_FLAG: Get the DMA Channel pending flags.
- __HAL_DMA_CLEAR_FLAG: Clear the DMA Channel pending flags.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Channel interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Channel interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Channel interrupt has occurred or not.

Note: You can refer to the DMA HAL driver header file for more useful macros

18.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Channel source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Channel priority value.

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual.
This section contains the following APIs:

- *HAL_DMA_Init*
- *HAL_DMA_DeInit*

18.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- *HAL_DMA_Start*
- *HAL_DMA_Start_IT*
- *HAL_DMA_Abort*
- *HAL_DMA_Abort_IT*
- *HAL_DMA_PollForTransfer*
- *HAL_DMA_IRQHandler*
- *HAL_DMA_RegisterCallback*
- *HAL_DMA_UnRegisterCallback*

18.2.4 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- *HAL_DMA_GetState*
- *HAL_DMA_GetError*

18.2.5 Detailed description of functions

HAL_DMA_Init

Function name

HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)

Function description

Initialize the DMA according to the specified parameters in the DMA_InitTypeDef and initialize the associated handle.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status

HAL_DMA_DeInit

Function name

HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)

Function description

DeInitialize the DMA peripheral.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status

HAL_DMA_Start

Function name

HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function description

Start the DMA Transfer.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination (up to 256Kbytes-1)

Return values

- **HAL:** status

HAL_DMA_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function description

Start the DMA Transfer with interrupt enabled.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination (up to 256Kbytes-1)

Return values

- **HAL:** status

HAL_DMA_Abort

Function name

HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)

Function description

Abort the DMA Transfer.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status

HAL_DMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)

Function description

Aborts the DMA Transfer in Interrupt mode.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status

HAL_DMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer complete.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CompleteLevel:** Specifies the DMA level complete.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_DMA_IRQHandler

Function name

void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)

Function description

Handle DMA interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **None:**

HAL_DMA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *_hdma) pCallback)

Function description

Register callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter.
- **pCallback:** pointer to private callbacsk function which has pointer to a DMA_HandleTypeDef structure as parameter.

Return values

- **HAL:** status

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

UnRegister callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)

Function description

Return the DMA hande state.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** state

HAL_DMA_GetError

Function name

uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)

Function description

Return the DMA error code.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** Error Code

18.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

18.3.1 DMA

DMA

DMA Data transfer direction

DMA_PERIPH_TO_MEMORY

Peripheral to memory direction

DMA_MEMORY_TO_PERIPH

Memory to peripheral direction

DMA_MEMORY_TO_MEMORY

Memory to memory direction

DMA Error Code

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_TE

Transfer error

HAL_DMA_ERROR_NO_XFER

Abort requested with no Xfer ongoing

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

HAL_DMA_ERROR_SYNC

DMAMUX sync overrun error

HAL_DMA_ERROR_REQGEN

DMAMUX request generator overrun error

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_GET_TC_FLAG_INDEX

Description:

- Return the current DMA Channel transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer complete flag index.

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- Return the current DMA Channel half transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified half transfer complete flag index.

__HAL_DMA_GET_TE_FLAG_INDEX

Description:

- Return the current DMA Channel transfer error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_GI_FLAG_INDEX

Description:

- Return the current DMA Channel Global interrupt flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_FLAG

Description:

- Get the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx Transfer complete flag
  - DMA_FLAG_HTx Half transfer complete flag
  - DMA_FLAG_TEx Transfer error flag
  - DMA_FLAG_GLx Global interrupt flag Where x can be from 1 to 8 to select the DMA Channel x flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clear the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx Transfer complete flag
  - DMA_FLAG_HTx Half transfer complete flag
  - DMA_FLAG_TEx Transfer error flag
  - DMA_FLAG_GLx Global interrupt flag Where x can be from 1 to 8 to select the DMA Channel x flag.

Return value:

- None

__HAL_DMA_ENABLE_IT

Description:

- Enable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC Transfer complete interrupt mask
  - DMA_IT_HT Half transfer complete interrupt mask
  - DMA_IT_TE Transfer error interrupt mask

Return value:

- None

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC Transfer complete interrupt mask
  - DMA_IT_HT Half transfer complete interrupt mask
  - DMA_IT_TE Transfer error interrupt mask

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Check whether the specified DMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - DMA_IT_TC Transfer complete interrupt mask
  - DMA_IT_HT Half transfer complete interrupt mask
  - DMA_IT_TE Transfer error interrupt mask

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_GET_COUNTER

Description:

- Return the number of remaining data units in the current DMA Channel transfer.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: number of remaining data units in the current DMA Channel transfer.

DMA flag definitions

DMA_FLAG_GL1

DMA_FLAG_TC1

DMA_FLAG_HT1

DMA_FLAG_TE1

DMA_FLAG_GL2

DMA_FLAG_TC2

DMA_FLAG_HT2

DMA_FLAG_TE2

DMA_FLAG_GL3

DMA_FLAG_TC3

DMA_FLAG_HT3

DMA_FLAG_TE3

DMA_FLAG_GL4

DMA_FLAG_TC4

DMA_FLAG_HT4

DMA_FLAG_TE4

DMA_FLAG_GL5

DMA_FLAG_TC5

DMA_FLAG_HT5

DMA_FLAG_TE5

DMA_FLAG_GL6

DMA_FLAG_TC6

DMA_FLAG_HT6

DMA_FLAG_TE6

DMA_FLAG_GL7

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

DMA_FLAG_GL8

DMA_FLAG_TC8

DMA_FLAG_HT8

DMA_FLAG_TE8

TIM DMA Handle Index

TIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3

Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4

Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_COMMUTATION

Index of the DMA handle used for Commutation DMA requests

TIM_DMA_ID_TRIGGER

Index of the DMA handle used for Trigger DMA requests

DMA interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE

Memory data alignment : Byte

DMA_MDATAALIGN_HALFWORD

Memory data alignment : HalfWord

DMA_MDATAALIGN_WORD

Memory data alignment : Word

DMA Memory incremented mode

DMA_MINC_ENABLE

Memory increment mode Enable

DMA_MINC_DISABLE

Memory increment mode Disable

DMA mode

DMA_NORMAL

Normal mode

DMA_CIRCULAR

Circular mode

DMA Peripheral data size

DMA_PDATAALIGN_BYTE

Peripheral data alignment : Byte

DMA_PDATAALIGN_HALFWORD

Peripheral data alignment : HalfWord

DMA_PDATAALIGN_WORD

Peripheral data alignment : Word

DMA Peripheral incremented mode

DMA_PINC_ENABLE

Peripheral increment mode Enable

DMA_PINC_DISABLE

Peripheral increment mode Disable

DMA Priority level

DMA_PRIORITY_LOW

Priority level : Low

DMA_PRIORITY_MEDIUM

Priority level : Medium

DMA_PRIORITY_HIGH

Priority level : High

DMA_PRIORITY_VERY_HIGH

Priority level : Very_High

DMA request

DMA_REQUEST_MEM2MEM

memory to memory transfer

DMA_REQUEST_GENERATOR0

DMA_REQUEST_GENERATOR1

DMA_REQUEST_GENERATOR2

DMA_REQUEST_GENERATOR3

DMA_REQUEST_ADC1

DMA_REQUEST_DAC1_CHANNEL1

DMA_REQUEST_DAC1_CHANNEL2

DMA_REQUEST_TIM6_UP

DMA_REQUEST_TIM7_UP

DMA_REQUEST_SPI1_RX

DMA_REQUEST_SPI1_TX

DMA_REQUEST_SPI2_RX

DMA_REQUEST_SPI2_TX

DMA_REQUEST_SPI3_RX

DMA_REQUEST_SPI3_TX

DMA_REQUEST_I2C1_RX

DMA_REQUEST_I2C1_TX

DMA_REQUEST_I2C2_RX

DMA_REQUEST_I2C2_TX

DMA_REQUEST_I2C3_RX

DMA_REQUEST_I2C3_TX

DMA_REQUEST_I2C4_RX

DMA_REQUEST_I2C4_TX

DMA_REQUEST_USART1_RX

DMA_REQUEST_USART1_TX

DMA_REQUEST_USART2_RX

DMA_REQUEST_USART2_TX

DMA_REQUEST_USART3_RX

DMA_REQUEST_USART3_TX

DMA_REQUEST_UART4_RX

DMA_REQUEST_UART4_TX

DMA_REQUEST_UART5_RX

DMA_REQUEST_UART5_TX

DMA_REQUEST_LPUART1_RX

DMA_REQUEST_LPUART1_TX

DMA_REQUEST_ADC2

DMA_REQUEST_ADC3

DMA_REQUEST_ADC4

DMA_REQUEST_ADC5

DMA_REQUEST_QUADSPI

DMA_REQUEST_DAC2_CHANNEL1

DMA_REQUEST_TIM1_CH1

DMA_REQUEST_TIM1_CH2

DMA_REQUEST_TIM1_CH3

DMA_REQUEST_TIM1_CH4

DMA_REQUEST_TIM1_UP

DMA_REQUEST_TIM1_TRIG

DMA_REQUEST_TIM1_COM

DMA_REQUEST_TIM8_CH1

DMA_REQUEST_TIM8_CH2

DMA_REQUEST_TIM8_CH3

DMA_REQUEST_TIM8_CH4

DMA_REQUEST_TIM8_UP

DMA_REQUEST_TIM8_TRIG

DMA_REQUEST_TIM8_COM

DMA_REQUEST_TIM2_CH1

DMA_REQUEST_TIM2_CH2

DMA_REQUEST_TIM2_CH3

DMA_REQUEST_TIM2_CH4

DMA_REQUEST_TIM2_UP

DMA_REQUEST_TIM3_CH1

DMA_REQUEST_TIM3_CH2

DMA_REQUEST_TIM3_CH3

DMA_REQUEST_TIM3_CH4

DMA_REQUEST_TIM3_UP

DMA_REQUEST_TIM3_TRIG

DMA_REQUEST_TIM4_CH1

DMA_REQUEST_TIM4_CH2

DMA_REQUEST_TIM4_CH3

DMA_REQUEST_TIM4_CH4

DMA_REQUEST_TIM4_UP

DMA_REQUEST_TIM5_CH1

DMA_REQUEST_TIM5_CH2

DMA_REQUEST_TIM5_CH3

DMA_REQUEST_TIM5_CH4

DMA_REQUEST_TIM5_UP

DMA_REQUEST_TIM5_TRIG

DMA_REQUEST_TIM15_CH1

DMA_REQUEST_TIM15_UP

DMA_REQUEST_TIM15_TRIG

DMA_REQUEST_TIM15_COM

DMA_REQUEST_TIM16_CH1

DMA_REQUEST_TIM16_UP

DMA_REQUEST_TIM17_CH1

DMA_REQUEST_TIM17_UP

DMA_REQUEST_TIM20_CH1

DMA_REQUEST_TIM20_CH2

DMA_REQUEST_TIM20_CH3

DMA_REQUEST_TIM20_CH4

DMA_REQUEST_TIM20_UP

DMA_REQUEST_AES_IN

DMA_REQUEST_AES_OUT

DMA_REQUEST_TIM20_TRIG

DMA_REQUEST_TIM20_COM

DMA_REQUEST_HRTIM1_M

DMA_REQUEST_HRTIM1_A

DMA_REQUEST_HRTIM1_B

DMA_REQUEST_HRTIM1_C

DMA_REQUEST_HRTIM1_D

DMA_REQUEST_HRTIM1_E

DMA_REQUEST_HRTIM1_F

DMA_REQUEST_DAC3_CHANNEL1

DMA_REQUEST_DAC3_CHANNEL2

DMA_REQUEST_DAC4_CHANNEL1

DMA_REQUEST_DAC4_CHANNEL2

DMA_REQUEST_SPI4_RX

DMA_REQUEST_SPI4_TX

DMA_REQUEST_SAI1_A

DMA_REQUEST_SAI1_B

DMA_REQUEST_FMAC_READ

DMA_REQUEST_FMAC_WRITE

DMA_REQUEST_CORDIC_READ

DMA_REQUEST_CORDIC_WRITE

DMA_REQUEST_UCPD1_RX

DMA_REQUEST_UCPD1_TX

19 HAL DMA Extension Driver

19.1 DMAEx Firmware driver registers structures

19.1.1 HAL_DMA_MuxSyncConfigTypeDef

HAL_DMA_MuxSyncConfigTypeDef is defined in the stm32g4xx_hal_dma_ex.h

Data Fields

- ***uint32_t SyncSignalID***
- ***uint32_t SyncPolarity***
- ***FunctionalState SyncEnable***
- ***FunctionalState EventEnable***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncSignalID***

  Specifies the synchronization signal gating the DMA request in periodic mode. This parameter can be a value of ***DMAEx_DMAMUX_SyncSignalID_selection***
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncPolarity***

  Specifies the polarity of the signal on which the DMA request is synchronized. This parameter can be a value of ***DMAEx_DMAMUX_SyncPolarity_selection***
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::SyncEnable***

  Specifies if the synchronization shall be enabled or disabled This parameter can take the value ENABLE or DISABLE
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::EventEnable***

  Specifies if an event shall be generated once the RequestNumber is reached. This parameter can take the value ENABLE or DISABLE
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::RequestNumber***

  Specifies the number of DMA request that will be authorized after a sync event This parameter must be a number between Min_Data = 1 and Max_Data = 32

19.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef

HAL_DMA_MuxRequestGeneratorConfigTypeDef is defined in the stm32g4xx_hal_dma_ex.h

Data Fields

- ***uint32_t SignalID***
- ***uint32_t Polarity***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::SignalID***

  Specifies the ID of the signal used for DMAMUX request generator This parameter can be a value of ***DMAEx_DMAMUX_SignalGeneratorID_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::Polarity***

  Specifies the polarity of the signal on which the request is generated. This parameter can be a value of ***DMAEx_DMAMUX_RequestGeneneratorPolarity_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::RequestNumber***

  Specifies the number of DMA request that will be generated after a signal event This parameter must be a number between Min_Data = 1 and Max_Data = 32

19.2 DMAEx Firmware driver API description

The following section lists the various functions of the DMAEx library.

19.2.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- To handle the DMAMUX Interrupts, the function HAL_DMAEx_MUX_IRQHandler should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler. As only one interrupt line is available for all DMAMUX channels and request generators , HAL_DMAEx_MUX_IRQHandler should be called with, as parameter, the appropriate DMA handle as many as used DMAs in the user project (exception done if a given DMA is not using the DMAMUX SYNC block neither a request generator)

19.2.2 Extended features functions

This section provides functions allowing to:

- Configure the DMAMUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMAMUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.

This section contains the following APIs:

- ***HAL_DMAEx_ConfigMuxSync***
- ***HAL_DMAEx_ConfigMuxRequestGenerator***
- ***HAL_DMAEx_EnableMuxRequestGenerator***
- ***HAL_DMAEx_DisableMuxRequestGenerator***
- ***HAL_DMAEx_MUX_IRQHandler***

19.2.3 Detailed description of functions

HAL_DMAEx_ConfigMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigMuxRequestGenerator (DMA_HandleTypeDef * hdma, HAL_DMA_MuxRequestGeneratorConfigTypeDef * pRequestGeneratorConfig)

Function description

Configure the DMAMUX request generator block used by the given DMA channel (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.
- **pRequestGeneratorConfig:** : pointer to HAL_DMA_MuxRequestGeneratorConfigTypeDef : contains the request generator parameters.

Return values

- **HAL:** status

HAL_DMAEx_EnableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_EnableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Enable the DMAMUX request generator block used by the given DMA channel (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status

HAL_DMAEx_DisableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_DisableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Disable the DMAMUX request generator block used by the given DMA channel (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status

HAL_DMAEx_ConfigMuxSync

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigMuxSync (DMA_HandleTypeDef * hdma, HAL_DMA_MuxSyncConfigTypeDef * pSyncConfig)

Function description

Configure the DMAMUX synchronization parameters for a given DMA channel (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.
- **pSyncConfig:** : pointer to HAL_DMA_MuxSyncConfigTypeDef : contains the DMAMUX synchronization parameters

Return values

- **HAL:** status

HAL_DMAEx_MUX_IRQHandler

Function name

void HAL_DMAEx_MUX_IRQHandler (DMA_HandleTypeDef * hdma)

Function description

Handles DMAMUX interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **None:**

19.3 DMAEx Firmware driver defines

The following section lists the various define and macros of the module.

19.3.1 DMAEx

DMAEx

DMAMUX RequestGeneneratorPolarity selection

HAL_DMAMUX_REQ_GEN_NO_EVENT

block request generator events

HAL_DMAMUX_REQ_GEN_RISING

generate request on rising edge events

HAL_DMAMUX_REQ_GEN_FALLING

generate request on falling edge events

HAL_DMAMUX_REQ_GEN_RISING_FALLING

generate request on rising and falling edge events

DMAMUX SignalGeneratorID selection

HAL_DMAMUX1_REQ_GEN_EXTI0

Request generator Signal is EXTI0 IT

HAL_DMAMUX1_REQ_GEN_EXTI1

Request generator Signal is EXTI1 IT

HAL_DMAMUX1_REQ_GEN_EXTI2

Request generator Signal is EXTI2 IT

HAL_DMAMUX1_REQ_GEN_EXTI3

Request generator Signal is EXTI3 IT

HAL_DMAMUX1_REQ_GEN_EXTI4

Request generator Signal is EXTI4 IT

HAL_DMAMUX1_REQ_GEN_EXTI5

Request generator Signal is EXTI5 IT

HAL_DMAMUX1_REQ_GEN_EXTI6

Request generator Signal is EXTI6 IT

HAL_DMAMUX1_REQ_GEN_EXTI7

Request generator Signal is EXTI7 IT

HAL_DMAMUX1_REQ_GEN_EXTI8

Request generator Signal is EXTI8 IT

HAL_DMAMUX1_REQ_GEN_EXTI9

Request generator Signal is EXTI9 IT

HAL_DMAMUX1_REQ_GEN_EXTI10

Request generator Signal is EXTI10 IT

HAL_DMAMUX1_REQ_GEN_EXTI11

Request generator Signal is EXTI11 IT

HAL_DMAMUX1_REQ_GEN_EXTI12

Request generator Signal is EXTI12 IT

HAL_DMAMUX1_REQ_GEN_EXTI13

Request generator Signal is EXTI13 IT

HAL_DMAMUX1_REQ_GEN_EXTI14

Request generator Signal is EXTI14 IT

HAL_DMAMUX1_REQ_GEN_EXTI15

Request generator Signal is EXTI15 IT

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH0_EVT

Request generator Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH1_EVT

Request generator Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH2_EVT

Request generator Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH3_EVT

Request generator Signal is DMAMUX1 Channel3 Event

HAL_DMAMUX1_REQ_GEN_LPTIM1_OUT

Request generator Signal is LPTIM1 OUT

DMAMUX SyncPolarity selection

HAL_DMAMUX_SYNC_NO_EVENT

block synchronization events

HAL_DMAMUX_SYNC_RISING

synchronize with rising edge events

HAL_DMAMUX_SYNC_FALLING

synchronize with falling edge events

HAL_DMAMUX_SYNC_RISING_FALLING

synchronize with rising and falling edge events

DMAMUX SyncSignalID selection

HAL_DMAMUX1_SYNC_EXTI0

Synchronization Signal is EXTI0 IT

HAL_DMAMUX1_SYNC_EXTI1

Synchronization Signal is EXTI1 IT

HAL_DMAMUX1_SYNC_EXTI2

Synchronization Signal is EXTI2 IT

HAL_DMAMUX1_SYNC_EXTI3

Synchronization Signal is EXTI3 IT

HAL_DMAMUX1_SYNC_EXTI4

Synchronization Signal is EXTI4 IT

HAL_DMAMUX1_SYNC_EXTI5

Synchronization Signal is EXTI5 IT

HAL_DMAMUX1_SYNC_EXTI6

Synchronization Signal is EXTI6 IT

HAL_DMAMUX1_SYNC_EXTI7

Synchronization Signal is EXTI7 IT

HAL_DMAMUX1_SYNC_EXTI8

Synchronization Signal is EXTI8 IT

HAL_DMAMUX1_SYNC_EXTI9

Synchronization Signal is EXTI9 IT

HAL_DMAMUX1_SYNC_EXTI10

Synchronization Signal is EXTI10 IT

HAL_DMAMUX1_SYNC_EXTI11

Synchronization Signal is EXTI11 IT

HAL_DMAMUX1_SYNC_EXTI12

Synchronization Signal is EXTI12 IT

HAL_DMAMUX1_SYNC_EXTI13

Synchronization Signal is EXTI13 IT

HAL_DMAMUX1_SYNC_EXTI14

Synchronization Signal is EXTI14 IT

HAL_DMAMUX1_SYNC_EXTI15

Synchronization Signal is EXTI15 IT

HAL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT

Synchronization Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT

Synchronization Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT

Synchronization Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH3_EVT

Synchronization Signal is DMAMUX1 Channel3 Event

HAL_DMAMUX1_SYNC_LPTIM1_OUT

Synchronization Signal is LPTIM1 OUT

20 HAL EXTI Generic Driver

20.1 EXTI Firmware driver registers structures

20.1.1 EXTI_HandleTypeDef

EXTI_HandleTypeDef is defined in the stm32g4xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***void(* PendingCallback***

Field Documentation

- ***uint32_t EXTI_HandleTypeDef::Line***
  Exti line number
- ***void(* EXTI_HandleTypeDef::PendingCallback)(void)***
  Exti pending callback

20.1.2 EXTI_ConfigTypeDef

EXTI_ConfigTypeDef is defined in the stm32g4xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***uint32_t Mode***
- ***uint32_t Trigger***
- ***uint32_t GPIOSel***

Field Documentation

- ***uint32_t EXTI_ConfigTypeDef::Line***
  The Exti line to be configured. This parameter can be a value of ***EXTI_Line***
- ***uint32_t EXTI_ConfigTypeDef::Mode***
  The Exit Mode to be configured for a core. This parameter can be a combination of ***EXTI_Mode***
- ***uint32_t EXTI_ConfigTypeDef::Trigger***
  The Exti Trigger to be configured. This parameter can be a value of ***EXTI_Trigger***
- ***uint32_t EXTI_ConfigTypeDef::GPIOSel***
  The Exti GPIO multiplexer selection to be configured. This parameter is only possible for line 0 to 15. It can be a value of ***EXTI_GPIOSel***

20.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

20.2.1 EXTI Peripheral features

- Each Exti line can be configured within this driver.
- Exti line can be configured in 3 different modes
  - Interrupt
  - Event
  - Both of them
- Configurable Exti lines can be configured with 3 different triggers
  - Rising
  - Falling
  - Both of them

- When set in interrupt mode, configurable Exti lines have two different interrupt pending registers which allow to distinguish which transition occurs:
  - Rising edge pending interrupt
  - Falling
- Exti lines 0 to 15 are linked to gpio pin number 0 to 15. Gpio port can be selected through multiplexer.

20.2.2 How to use this driver

1. Configure the EXTI line using HAL_EXTI_SetConfigLine().
   - Choose the interrupt line number by setting "Line" member from EXTI_ConfigTypeDef structure.
   - Configure the interrupt and/or event mode using "Mode" member from EXTI_ConfigTypeDef structure.
   - For configurable lines, configure rising and/or falling trigger "Trigger" member from EXTI_ConfigTypeDef structure.
   - For Exti lines linked to gpio, choose gpio port using "GPIOSel" member from GPIO_InitTypeDef structure.
2. Get current Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
   - Provide pointer on EXTI_ConfigTypeDef structure as second parameter.
3. Clear Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
4. Register callback to treat Exti interrupts using HAL_EXTI_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from EXTI_CallbackIDTypeDef.
   - Provide callback function pointer.
5. Get interrupt pending bit using HAL_EXTI_GetPending().
6. Clear interrupt pending bit using HAL_EXTI_ClearPending().
7. Generate software interrupt using HAL_EXTI_GenerateSWI().

20.2.3 Configuration functions

This section contains the following APIs:

- ***HAL_EXTI_SetConfigLine***
- ***HAL_EXTI_GetConfigLine***
- ***HAL_EXTI_ClearConfigLine***
- ***HAL_EXTI_RegisterCallback***
- ***HAL_EXTI_GetHandle***

20.2.4 Detailed description of functions

HAL_EXTI_SetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_SetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Set configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on EXTI configuration to be set.

Return values

- **HAL:** Status.

HAL_EXTI_GetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Get configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on structure to store Exti configuration.

Return values

- **HAL:** Status.

HAL_EXTI_ClearConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_ClearConfigLine (EXTI_HandleTypeDef * hexti)

Function description

Clear whole configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.

Return values

- **HAL:** Status.

HAL_EXTI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_EXTI_RegisterCallback (EXTI_HandleTypeDef * hexti, EXTI_CallbackIDTypeDef CallbackID, void(*)(void) pPendingCbfn)

Function description

Register callback for a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **CallbackID:** User callback identifier. This parameter can be one of
  - EXTI_CallbackIDTypeDef values.
- **pPendingCbfn:** function pointer to be stored as callback.

Return values

- **HAL:** Status.

HAL_EXTI_GetHandle

Function name

HAL_StatusTypeDef HAL_EXTI_GetHandle (EXTI_HandleTypeDef * hexti, uint32_t ExtiLine)

Function description

Store line number as handle private field.

Parameters

- **hexti:** Exti handle.
- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.

Return values

- **HAL:** Status.

HAL_EXTI_IRQHandler

Function name

void HAL_EXTI_IRQHandler (EXTI_HandleTypeDef * hexti)

Function description

Handle EXTI interrupt request.

Parameters

- **hexti:** Exti handle.

Return values

- **none.:**

HAL_EXTI_GetPending

Function name

uint32_t HAL_EXTI_GetPending (EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Get interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** unused

Return values

- **1:** if interrupt is pending else 0.

HAL_EXTI_ClearPending

Function name

void HAL_EXTI_ClearPending (EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Clear interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** unused

Return values

- **None.:**

HAL_EXTI_GenerateSWI

Function name

void HAL_EXTI_GenerateSWI (EXTI_HandleTypeDef * hexti)

Function description

Generate a software interrupt for a dedicated line.

Parameters

- **hexti:** Exti handle.

Return values

- **None.:**

20.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

20.3.1 EXTI

EXTI

EXTI GPIOSel

EXTI_GPIOA

EXTI_GPIOB

EXTI_GPIOC

EXTI_GPIOD

EXTI_GPIOE

EXTI_GPIOF

EXTI_GPIOG

EXTI Line

EXTI_LINE_0

EXTI_LINE_1

EXTI_LINE_2

EXTI_LINE_3

EXTI_LINE_4

EXTI_LINE_5

EXTI_LINE_6

EXTI_LINE_7

EXTI_LINE_8

EXTI_LINE_9

EXTI_LINE_10

EXTI_LINE_11

EXTI_LINE_12

EXTI_LINE_13

EXTI_LINE_14

EXTI_LINE_15

EXTI_LINE_16

EXTI_LINE_17

EXTI_LINE_18

EXTI_LINE_19

EXTI_LINE_20

EXTI_LINE_21

EXTI_LINE_22

EXTI_LINE_23

EXTI_LINE_24

EXTI_LINE_25

EXTI_LINE_26

EXTI_LINE_27

EXTI_LINE_28

EXTI_LINE_29

EXTI_LINE_30

EXTI_LINE_31

EXTI_LINE_32

EXTI_LINE_33

EXTI_LINE_34

EXTI_LINE_35

EXTI_LINE_36

EXTI_LINE_37

EXTI_LINE_38

EXTI_LINE_39

EXTI_LINE_40

EXTI_LINE_41

EXTI_LINE_42

EXTI_LINE_43

EXTI Mode

EXTI_MODE_NONE

EXTI_MODE_INTERRUPT

EXTI_MODE_EVENT

EXTI Trigger

EXTI_TRIGGER_NONE

EXTI_TRIGGER_RISING

EXTI_TRIGGER_FALLING

EXTI_TRIGGER_RISING_FALLING

21 HAL FDCAN Generic Driver

21.1 FDCAN Firmware driver registers structures

21.1.1 FDCAN_InitTypeDef

FDCAN_InitTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t ClockDivider***
- ***uint32_t FrameFormat***
- ***uint32_t Mode***
- ***FunctionalState AutoRetransmission***
- ***FunctionalState TransmitPause***
- ***FunctionalState ProtocolException***
- ***uint32_t NominalPrescaler***
- ***uint32_t NominalSyncJumpWidth***
- ***uint32_t NominalTimeSeg1***
- ***uint32_t NominalTimeSeg2***
- ***uint32_t DataPrescaler***
- ***uint32_t DataSyncJumpWidth***
- ***uint32_t DataTimeSeg1***
- ***uint32_t DataTimeSeg2***
- ***uint32_t StdFiltersNbr***
- ***uint32_t ExtFiltersNbr***
- ***uint32_t TxFifoQueueMode***

Field Documentation

- ***uint32_t FDCAN_InitTypeDef::ClockDivider***

  Specifies the FDCAN kernel clock divider. The clock is common to all FDCAN instances. This parameter is applied only at initialisation of first FDCAN instance. This parameter can be a value of ***FDCAN_clock_divider***.
- ***uint32_t FDCAN_InitTypeDef::FrameFormat***

  Specifies the FDCAN frame format. This parameter can be a value of ***FDCAN_frame_format***
- ***uint32_t FDCAN_InitTypeDef::Mode***

  Specifies the FDCAN mode. This parameter can be a value of ***FDCAN_operating_mode***
- ***FunctionalState FDCAN_InitTypeDef::AutoRetransmission***

  Enable or disable the automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::TransmitPause***

  Enable or disable the Transmit Pause feature. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::ProtocolException***

  Enable or disable the Protocol Exception Handling. This parameter can be set to ENABLE or DISABLE
- ***uint32_t FDCAN_InitTypeDef::NominalPrescaler***

  Specifies the value by which the oscillator frequency is divided for generating the nominal bit time quanta. This parameter must be a number between 1 and 512
- ***uint32_t FDCAN_InitTypeDef::NominalSyncJumpWidth***

  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter must be a number between 1 and 128
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg1***

  Specifies the number of time quanta in Bit Segment 1. This parameter must be a number between 2 and 256

- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg2***
  Specifies the number of time quanta in Bit Segment 2. This parameter must be a number between 2 and 128
- ***uint32_t FDCAN_InitTypeDef::DataPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the data bit time quanta. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a data bit to perform resynchronization. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg1***
  Specifies the number of time quanta in Data Bit Segment 1. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg2***
  Specifies the number of time quanta in Data Bit Segment 2. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::StdFiltersNbr***
  Specifies the number of standard Message ID filters. This parameter must be a number between 0 and 28
- ***uint32_t FDCAN_InitTypeDef::ExtFiltersNbr***
  Specifies the number of extended Message ID filters. This parameter must be a number between 0 and 8
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueMode***
  Tx FIFO/Queue Mode selection. This parameter can be a value of ***FDCAN_txFifoQueue_Mode***

21.1.2 FDCAN_FilterTypeDef

FDCAN_FilterTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t IdType***
- ***uint32_t FilterIndex***
- ***uint32_t FilterType***
- ***uint32_t FilterConfig***
- ***uint32_t FilterID1***
- ***uint32_t FilterID2***

Field Documentation

- ***uint32_t FDCAN_FilterTypeDef::IdType***
  Specifies the identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_FilterTypeDef::FilterIndex***
  Specifies the filter which will be initialized. This parameter must be a number between:
  – 0 and (SRAMCAN_FLS_NBR-1), if IdType is FDCAN_STANDARD_ID
  – 0 and (SRAMCAN_FLE_NBR-1), if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterType***
  Specifies the filter type. This parameter can be a value of ***FDCAN_filter_type***. The value FDCAN_FILTER_RANGE_NO_EIDM is permitted only when IdType is FDCAN_EXTENDED_ID.
- ***uint32_t FDCAN_FilterTypeDef::FilterConfig***
  Specifies the filter configuration. This parameter can be a value of ***FDCAN_filter_config***
- ***uint32_t FDCAN_FilterTypeDef::FilterID1***
  Specifies the filter identification 1. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterID2***
  Specifies the filter identification 2. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID

21.1.3 FDCAN_TxHeaderTypeDef

FDCAN_TxHeaderTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxEventFifoControl***
- ***uint32_t MessageMarker***

Field Documentation

- ***uint32_t FDCAN_TxHeaderTypeDef::Identifier***

  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxHeaderTypeDef::IdType***

  Specifies the identifier type for the message that will be transmitted. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxFrameType***

  Specifies the frame type of the message that will be transmitted. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::DataLength***

  Specifies the length of the frame that will be transmitted. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxHeaderTypeDef::ErrorStateIndicator***

  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxHeaderTypeDef::BitRateSwitch***

  Specifies whether the Tx frame will be transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxHeaderTypeDef::FDFormat***

  Specifies whether the Tx frame will be transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxEventFifoControl***

  Specifies the event FIFO control. This parameter can be a value of ***Section 21.3.1 FDCAN_EFC***
- ***uint32_t FDCAN_TxHeaderTypeDef::MessageMarker***

  Specifies the message marker to be copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF

21.1.4 FDCAN_RxHeaderTypeDef

FDCAN_RxHeaderTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t RxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***

- ***uint32_t RxTimestamp***
- ***uint32_t FilterIndex***
- ***uint32_t IsFilterMatchingFrame***

Field Documentation

- ***uint32_t FDCAN_RxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_RxHeaderTypeDef::IdType***
  Specifies the identifier type of the received message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxFrameType***
  Specifies the the received message frame type. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::DataLength***
  Specifies the received frame length. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_RxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_RxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Rx frame is received with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_RxHeaderTypeDef::FDFormat***
  Specifies whether the Rx frame is received in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxTimestamp***
  Specifies the timestamp counter value captured on start of frame reception. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_RxHeaderTypeDef::FilterIndex***
  Specifies the index of matching Rx acceptance filter element. This parameter must be a number between:
  – 0 and (SRAMCAN_FLS_NBR-1), if IdType is FDCAN_STANDARD_ID
  – 0 and (SRAMCAN_FLE_NBR-1), if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_RxHeaderTypeDef::IsFilterMatchingFrame***
  Specifies whether the accepted frame did not match any Rx filter. Acceptance of non-matching frames may be enabled via **HAL_FDCAN_ConfigGlobalFilter()**. This parameter can be 0 or 1

21.1.5 FDCAN_TxEventFifoTypeDef

FDCAN_TxEventFifoTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxTimestamp***
- ***uint32_t MessageMarker***
- ***uint32_t EventType***

Field Documentation

- ***uint32_t FDCAN_TxEventFifoTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxEventFifoTypeDef::IdType***
  Specifies the identifier type for the transmitted message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxFrameType***
  Specifies the frame type of the transmitted message. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::DataLength***
  Specifies the length of the transmitted frame. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxEventFifoTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxEventFifoTypeDef::BitRateSwitch***
  Specifies whether the Tx frame is transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxEventFifoTypeDef::FDFormat***
  Specifies whether the Tx frame is transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxTimestamp***
  Specifies the timestamp counter value captured on start of frame transmission. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::MessageMarker***
  Specifies the message marker copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::EventType***
  Specifies the event type. This parameter can be a value of ***FDCAN_event_type***

21.1.6 FDCAN_HpMsgStatusTypeDef

FDCAN_HpMsgStatusTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t FilterList***
- ***uint32_t FilterIndex***
- ***uint32_t MessageStorage***
- ***uint32_t MessageIndex***

Field Documentation

- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterList***
  Specifies the filter list of the matching filter element. This parameter can be:
  – 0 : Standard Filter List
  – 1 : Extended Filter List
- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterIndex***
  Specifies the index of matching filter element. This parameter can be a number between:
  – 0 and (SRAMCAN_FLS_NBR-1), if FilterList is 0 (Standard)
  – 0 and (SRAMCAN_FLE_NBR-1), if FilterList is 1 (Extended)
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageStorage***
  Specifies the HP Message Storage. This parameter can be a value of ***FDCAN_hp_msg_storage***
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageIndex***
  Specifies the Index of Rx FIFO element to which the message was stored. This parameter is valid only when MessageStorage is: FDCAN_HP_STORAGE_RXFIFO0 or FDCAN_HP_STORAGE_RXFIFO1

21.1.7 FDCAN_ProtocolStatusTypeDef

FDCAN_ProtocolStatusTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t LastErrorCode***
- ***uint32_t DataLastErrorCode***
- ***uint32_t Activity***
- ***uint32_t ErrorPassive***
- ***uint32_t Warning***
- ***uint32_t BusOff***
- ***uint32_t RxESIflag***
- ***uint32_t RxBRSflag***
- ***uint32_t RxFDFflag***
- ***uint32_t ProtocolException***
- ***uint32_t TDCvalue***

Field Documentation

- ***uint32_t FDCAN_ProtocolStatusTypeDef::LastErrorCode***

  Specifies the type of the last error that occurred on the FDCAN bus. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::DataLastErrorCode***

  Specifies the type of the last error that occurred in the data phase of a CAN FD format frame with its BRS flag set. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Activity***

  Specifies the FDCAN module communication state. This parameter can be a value of ***FDCAN_communication_state***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ErrorPassive***

  Specifies the FDCAN module error status. This parameter can be:
  - 0 : The FDCAN is in Error_Active state
  - 1 : The FDCAN is in Error_Passive state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Warning***

  Specifies the FDCAN module warning status. This parameter can be:
  - 0 : error counters (RxErrorCnt and TxErrorCnt) are below the Error_Warning limit of 96
  - 1 : at least one of error counters has reached the Error_Warning limit of 96
- ***uint32_t FDCAN_ProtocolStatusTypeDef::BusOff***

  Specifies the FDCAN module Bus_Off status. This parameter can be:
  - 0 : The FDCAN is not in Bus_Off state
  - 1 : The FDCAN is in Bus_Off state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxESIflag***

  Specifies ESI flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its ESI flag set
  - 1 : Last received CAN FD message had its ESI flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxBRSflag***

  Specifies BRS flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its BRS flag set
  - 1 : Last received CAN FD message had its BRS flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxFDFflag***

  Specifies if CAN FD message (FDF flag set) has been received since last protocol status This parameter can be:
  - 0 : No CAN FD message received
  - 1 : CAN FD message received

- ***uint32_t FDCAN_ProtocolStatusTypeDef::ProtocolException***
  Specifies the FDCAN module Protocol Exception status. This parameter can be:
  – 0 : No protocol exception event occurred since last read access
  – 1 : Protocol exception event occurred
- ***uint32_t FDCAN_ProtocolStatusTypeDef::TDCvalue***
  Specifies the Transmitter Delay Compensation Value. This parameter can be a number between 0 and 127

21.1.8 FDCAN_ErrorCountersTypeDef

FDCAN_ErrorCountersTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t TxErrorCnt***
- ***uint32_t RxErrorCnt***
- ***uint32_t RxErrorPassive***
- ***uint32_t ErrorLogging***

Field Documentation

- ***uint32_t FDCAN_ErrorCountersTypeDef::TxErrorCnt***
  Specifies the Transmit Error Counter Value. This parameter can be a number between 0 and 255
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorCnt***
  Specifies the Receive Error Counter Value. This parameter can be a number between 0 and 127
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorPassive***
  Specifies the Receive Error Passive status. This parameter can be:
  – 0 : The Receive Error Counter (RxErrorCnt) is below the error passive level of 128
  – 1 : The Receive Error Counter (RxErrorCnt) has reached the error passive level of 128
- ***uint32_t FDCAN_ErrorCountersTypeDef::ErrorLogging***
  Specifies the Transmit/Receive error logging counter value. This parameter can be a number between 0 and 255. This counter is incremented each time when a FDCAN protocol error causes the TxErrorCnt or the RxErrorCnt to be incremented. The counter stops at 255; the next increment of TxErrorCnt or RxErrorCnt sets interrupt flag FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

21.1.9 FDCAN_MsgRamAddressTypeDef

FDCAN_MsgRamAddressTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***uint32_t StandardFilterSA***
- ***uint32_t ExtendedFilterSA***
- ***uint32_t RxFIFO0SA***
- ***uint32_t RxFIFO1SA***
- ***uint32_t TxEventFIFOSA***
- ***uint32_t TxFIFOQSA***

Field Documentation

- ***uint32_t FDCAN_MsgRamAddressTypeDef::StandardFilterSA***
  Specifies the Standard Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::ExtendedFilterSA***
  Specifies the Extended Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO0SA***
  Specifies the Rx FIFO 0 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO1SA***
  Specifies the Rx FIFO 1 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxEventFIFOSA***
  Specifies the Tx Event FIFO Start Address. This parameter must be a 32-bit word address

- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxFIFOQSA***
  Specifies the Tx FIFO/Queue Start Address. This parameter must be a 32-bit word address

21.1.10 FDCAN_HandleTypeDef

FDCAN_HandleTypeDef is defined in the stm32g4xx_hal_fdcan.h

Data Fields

- ***FDCAN_GlobalTypeDef * Instance***
- ***FDCAN_InitTypeDef Init***
- ***FDCAN_MsgRamAddressTypeDef msgRam***
- ***uint32_t LatestTxFifoQRequest***
- ***__IO HAL_FDCAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***FDCAN_GlobalTypeDef* FDCAN_HandleTypeDef::Instance***
  Register base address
- ***FDCAN_InitTypeDef FDCAN_HandleTypeDef::Init***
  FDCAN required parameters
- ***FDCAN_MsgRamAddressTypeDef FDCAN_HandleTypeDef::msgRam***
  FDCAN Message RAM blocks
- ***uint32_t FDCAN_HandleTypeDef::LatestTxFifoQRequest***
  FDCAN Tx buffer index of latest Tx FIFO/Queue request
- ***__IO HAL_FDCAN_StateTypeDef FDCAN_HandleTypeDef::State***
  FDCAN communication state
- ***HAL_LockTypeDef FDCAN_HandleTypeDef::Lock***
  FDCAN locking object
- ***__IO uint32_t FDCAN_HandleTypeDef::ErrorCode***
  FDCAN Error code

21.2 FDCAN Firmware driver API description

The following section lists the various functions of the FDCAN library.

21.2.1 How to use this driver

1. Initialize the FDCAN peripheral using HAL_FDCAN_Init function.

2. If needed , configure the reception filters and optional features using the following configuration functions:
   – HAL_FDCAN_ConfigFilter
   – HAL_FDCAN_ConfigGlobalFilter
   – HAL_FDCAN_ConfigExtendedIdMask
   – HAL_FDCAN_ConfigRxFifoOverwrite
   – HAL_FDCAN_ConfigRamWatchdog
   – HAL_FDCAN_ConfigTimestampCounter
   – HAL_FDCAN_EnableTimestampCounter
   – HAL_FDCAN_DisableTimestampCounter
   – HAL_FDCAN_ConfigTimeoutCounter
   – HAL_FDCAN_EnableTimeoutCounter
   – HAL_FDCAN_DisableTimeoutCounter
   – HAL_FDCAN_ConfigTxDelayCompensation
   – HAL_FDCAN_EnableTxDelayCompensation
   – HAL_FDCAN_DisableTxDelayCompensation
   – HAL_FDCAN_EnableISOMode
   – HAL_FDCAN_DisableISOMode
   – HAL_FDCAN_EnableEdgeFiltering
   – HAL_FDCAN_DisableEdgeFiltering
3. Start the FDCAN module using HAL_FDCAN_Start function. At this level the node is active on the bus: it can send and receive messages.
4. The following Tx control functions can only be called when the FDCAN module is started:
   – HAL_FDCAN_AddMessageToTxFifoQ
   – HAL_FDCAN_AbortTxRequest
5. After having submitted a Tx request in Tx Fifo or Queue, it is possible to get Tx buffer location used to place the Tx request thanks to HAL_FDCAN_GetLatestTxFifoQRequestBuffer API. It is then possible to abort later on the corresponding Tx Request using HAL_FDCAN_AbortTxRequest API.
6. When a message is received into the FDCAN message RAM, it can be retrieved using the HAL_FDCAN_GetRxMessage function.
7. Calling the HAL_FDCAN_Stop function stops the FDCAN module by entering it to initialization mode and re-enabling access to configuration registers through the configuration functions listed here above.
8. All other control functions can be called any time after initialization phase, no matter if the FDCAN module is started or stopped.

Polling mode operation

1. Reception and transmission states can be monitored via the following functions:
   – HAL_FDCAN_IsTxBufferMessagePending
   – HAL_FDCAN_GetRxFifoFillLevel
   – HAL_FDCAN_GetTxFifoFreeLevel

Interrupt mode operation

1. There are two interrupt lines: line 0 and 1. By default, all interrupts are assigned to line 0. Interrupt lines can be configured using HAL_FDCAN_ConfigInterruptLines function.
2. Notifications are activated using HAL_FDCAN_ActivateNotification function. Then, the process can be controlled through one of the available user callbacks: HAL_FDCAN_xxxCallback.

Callback registration

21.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the FDCAN.

- De-initialize the FDCAN.
- Enter FDCAN peripheral in power down mode.
- Exit power down mode.
- Register callbacks.
- Unregister callbacks.

This section contains the following APIs:

- ***HAL_FDCAN_Init***
- ***HAL_FDCAN_DeInit***
- ***HAL_FDCAN_MspInit***
- ***HAL_FDCAN_MspDeInit***
- ***HAL_FDCAN_EnterPowerDownMode***
- ***HAL_FDCAN_ExitPowerDownMode***

21.2.3 Configuration functions

This section provides functions allowing to:

- HAL_FDCAN_ConfigFilter : Configure the FDCAN reception filters
- HAL_FDCAN_ConfigGlobalFilter : Configure the FDCAN global filter
- HAL_FDCAN_ConfigExtendedIdMask : Configure the extended ID mask
- HAL_FDCAN_ConfigRxFifoOverwrite : Configure the Rx FIFO operation mode
- HAL_FDCAN_ConfigRamWatchdog : Configure the RAM watchdog
- HAL_FDCAN_ConfigTimestampCounter : Configure the timestamp counter
- HAL_FDCAN_EnableTimestampCounter : Enable the timestamp counter
- HAL_FDCAN_DisableTimestampCounter : Disable the timestamp counter
- HAL_FDCAN_GetTimestampCounter : Get the timestamp counter value
- HAL_FDCAN_ResetTimestampCounter : Reset the timestamp counter to zero
- HAL_FDCAN_ConfigTimeoutCounter : Configure the timeout counter
- HAL_FDCAN_EnableTimeoutCounter : Enable the timeout counter
- HAL_FDCAN_DisableTimeoutCounter : Disable the timeout counter
- HAL_FDCAN_GetTimeoutCounter : Get the timeout counter value
- HAL_FDCAN_ResetTimeoutCounter : Reset the timeout counter to its start value
- HAL_FDCAN_ConfigTxDelayCompensation : Configure the transmitter delay compensation
- HAL_FDCAN_EnableTxDelayCompensation : Enable the transmitter delay compensation
- HAL_FDCAN_DisableTxDelayCompensation : Disable the transmitter delay compensation
- HAL_FDCAN_EnableISOMode : Enable ISO 11898-1 protocol mode
- HAL_FDCAN_DisableISOMode : Disable ISO 11898-1 protocol mode
- HAL_FDCAN_EnableEdgeFiltering : Enable edge filtering during bus integration
- HAL_FDCAN_DisableEdgeFiltering : Disable edge filtering during bus integration

This section contains the following APIs:

- ***HAL_FDCAN_ConfigFilter***
- ***HAL_FDCAN_ConfigGlobalFilter***
- ***HAL_FDCAN_ConfigExtendedIdMask***
- ***HAL_FDCAN_ConfigRxFifoOverwrite***
- ***HAL_FDCAN_ConfigRamWatchdog***
- ***HAL_FDCAN_ConfigTimestampCounter***
- ***HAL_FDCAN_EnableTimestampCounter***
- ***HAL_FDCAN_DisableTimestampCounter***
- ***HAL_FDCAN_GetTimestampCounter***
- ***HAL_FDCAN_ResetTimestampCounter***
- ***HAL_FDCAN_ConfigTimeoutCounter***

- *HAL_FDCAN_EnableTimeoutCounter*
- *HAL_FDCAN_DisableTimeoutCounter*
- *HAL_FDCAN_GetTimeoutCounter*
- *HAL_FDCAN_ResetTimeoutCounter*
- *HAL_FDCAN_ConfigTxDelayCompensation*
- *HAL_FDCAN_EnableTxDelayCompensation*
- *HAL_FDCAN_DisableTxDelayCompensation*
- *HAL_FDCAN_EnableISOMode*
- *HAL_FDCAN_DisableISOMode*
- *HAL_FDCAN_EnableEdgeFiltering*
- *HAL_FDCAN_DisableEdgeFiltering*

21.2.4 Control functions

This section provides functions allowing to:

- HAL_FDCAN_Start : Start the FDCAN module
- HAL_FDCAN_Stop : Stop the FDCAN module and enable access to configuration registers
- HAL_FDCAN_AddMessageToTxFifoQ : Add a message to the Tx FIFO/Queue and activate the corresponding transmission request
- HAL_FDCAN_GetLatestTxFifoQRequestBuffer : Get Tx buffer index of latest Tx FIFO/Queue request
- HAL_FDCAN_AbortTxRequest : Abort transmission request
- HAL_FDCAN_GetRxMessage : Get an FDCAN frame from the Rx FIFO zone into the message RAM
- HAL_FDCAN_GetTxEvent : Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM
- HAL_FDCAN_GetHighPriorityMessageStatus : Get high priority message status
- HAL_FDCAN_GetProtocolStatus : Get protocol status
- HAL_FDCAN_GetErrorCounters : Get error counter values
- HAL_FDCAN_IsTxBufferMessagePending : Check if a transmission request is pending on the selected Tx buffer
- HAL_FDCAN_GetRxFifoFillLevel : Return Rx FIFO fill level
- HAL_FDCAN_GetTxFifoFreeLevel : Return Tx FIFO free level
- HAL_FDCAN_IsRestrictedOperationMode : Check if the FDCAN peripheral entered Restricted Operation Mode
- HAL_FDCAN_ExitRestrictedOperationMode : Exit Restricted Operation Mode

This section contains the following APIs:

- *HAL_FDCAN_Start*
- *HAL_FDCAN_Stop*
- *HAL_FDCAN_AddMessageToTxFifoQ*
- *HAL_FDCAN_GetLatestTxFifoQRequestBuffer*
- *HAL_FDCAN_AbortTxRequest*
- *HAL_FDCAN_GetRxMessage*
- *HAL_FDCAN_GetTxEvent*
- *HAL_FDCAN_GetHighPriorityMessageStatus*
- *HAL_FDCAN_GetProtocolStatus*
- *HAL_FDCAN_GetErrorCounters*
- *HAL_FDCAN_IsTxBufferMessagePending*
- *HAL_FDCAN_GetRxFifoFillLevel*
- *HAL_FDCAN_GetTxFifoFreeLevel*
- *HAL_FDCAN_IsRestrictedOperationMode*
- *HAL_FDCAN_ExitRestrictedOperationMode*

21.2.5 Interrupts management

This section provides functions allowing to:

- HAL_FDCAN_ConfigInterruptLines : Assign interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_ActivateNotification : Enable interrupts
- HAL_FDCAN_DeactivateNotification : Disable interrupts
- HAL_FDCAN_IRQHandler : Handles FDCAN interrupt request

This section contains the following APIs:

- ***HAL_FDCAN_ConfigInterruptLines***
- ***HAL_FDCAN_ActivateNotification***
- ***HAL_FDCAN_DeactivateNotification***
- ***HAL_FDCAN_IRQHandler***

21.2.6 Callback functions

This subsection provides the following callback functions:

- HAL_FDCAN_TxEventFifoCallback
- HAL_FDCAN_RxFifo0Callback
- HAL_FDCAN_RxFifo1Callback
- HAL_FDCAN_TxFifoEmptyCallback
- HAL_FDCAN_TxBufferCompleteCallback
- HAL_FDCAN_TxBufferAbortCallback
- HAL_FDCAN_HighPriorityMessageCallback
- HAL_FDCAN_TimestampWraparoundCallback
- HAL_FDCAN_TimeoutOccurredCallback
- HAL_FDCAN_ErrorCallback
- HAL_FDCAN_ErrorStatusCallback

This section contains the following APIs:

- ***HAL_FDCAN_TxEventFifoCallback***
- ***HAL_FDCAN_RxFifo0Callback***
- ***HAL_FDCAN_RxFifo1Callback***
- ***HAL_FDCAN_TxFifoEmptyCallback***
- ***HAL_FDCAN_TxBufferCompleteCallback***
- ***HAL_FDCAN_TxBufferAbortCallback***
- ***HAL_FDCAN_TimestampWraparoundCallback***
- ***HAL_FDCAN_TimeoutOccurredCallback***
- ***HAL_FDCAN_HighPriorityMessageCallback***
- ***HAL_FDCAN_ErrorCallback***
- ***HAL_FDCAN_ErrorStatusCallback***

21.2.7 Peripheral State functions

This subsection provides functions allowing to :

- HAL_FDCAN_GetState() : Return the FDCAN state.
- HAL_FDCAN_GetError() : Return the FDCAN error code if any.

This section contains the following APIs:

- ***HAL_FDCAN_GetState***
- ***HAL_FDCAN_GetError***

21.2.8 Detailed description of functions

HAL_FDCAN_Init

Function name

HAL_StatusTypeDef HAL_FDCAN_Init (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN peripheral according to the specified parameters in the FDCAN_InitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DeInit

Function name

HAL_StatusTypeDef HAL_FDCAN_DeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Deinitializes the FDCAN peripheral registers to their default reset values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_MspInit

Function name

void HAL_FDCAN_MspInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_MspDeInit

Function name

void HAL_FDCAN_MspDeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

DeInitializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_EnterPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnterPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enter FDCAN peripheral in sleep mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ExitPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit power down mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFilter (FDCAN_HandleTypeDef * hfdcan, FDCAN_FilterTypeDef * sFilterConfig)

Function description

Configure the FDCAN reception filter according to the specified parameters in the FDCAN_FilterTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sFilterConfig:** pointer to an FDCAN_FilterTypeDef structure that contains the filter configuration information

Return values

- **HAL:** status

HAL_FDCAN_ConfigGlobalFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigGlobalFilter (FDCAN_HandleTypeDef * hfdcan, uint32_t NonMatchingStd, uint32_t NonMatchingExt, uint32_t RejectRemoteStd, uint32_t RejectRemoteExt)

Function description

Configure the FDCAN global filter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NonMatchingStd:** Defines how received messages with 11-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  - FDCAN_Non_Matching_Frames.
- **NonMatchingExt:** Defines how received messages with 29-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  - FDCAN_Non_Matching_Frames.
- **RejectRemoteStd:** Filter or reject all the remote 11-bit IDs frames. This parameter can be a value of
  - FDCAN_Reject_Remote_Frames.
- **RejectRemoteExt:** Filter or reject all the remote 29-bit IDs frames. This parameter can be a value of
  - FDCAN_Reject_Remote_Frames.

Return values

- **HAL:** status

HAL_FDCAN_ConfigExtendedIdMask

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigExtendedIdMask (FDCAN_HandleTypeDef * hfdcan, uint32_t Mask)

Function description

Configure the extended ID mask.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Mask:** Extended ID Mask. This parameter must be a number between 0 and 0x1FFFFFFF

Return values

- **HAL:** status

HAL_FDCAN_ConfigRxFifoOverwrite

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRxFifoOverwrite (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo, uint32_t OperationMode)

Function description

Configure the Rx FIFO operation mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1
- **OperationMode:** operation mode. This parameter can be a value of
  - FDCAN_Rx_FIFO_operation_mode.

Return values

- **HAL:** status

HAL_FDCAN_ConfigRamWatchdog

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRamWatchdog (FDCAN_HandleTypeDef * hfdcan, uint32_t CounterStartValue)

Function description

Configure the RAM watchdog.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **CounterStartValue:** Start value of the Message RAM Watchdog Counter, This parameter must be a number between 0x00 and 0xFF, with the reset value of 0x00 the counter is disabled.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampPrescaler)

Function description

Configure the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampPrescaler:** Timestamp Counter Prescaler. This parameter can be a value of
  - FDCAN_Timestamp_Prescaler.

Return values

- **HAL:** status

HAL_FDCAN_EnableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampOperation)

Function description

Enable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampOperation:** Timestamp counter operation. This parameter can be a value of
  - FDCAN_Timestamp.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimestampCounter

Function name

uint16_t HAL_FDCAN_GetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timestamp counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Timestamp:** counter value

HAL_FDCAN_ResetTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timestamp counter to zero.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimeoutCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeoutOperation, uint32_t TimeoutPeriod)

Function description

Configure the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeoutOperation:** Timeout counter operation. This parameter can be a value of
  – FDCAN_Timeout_Operation.
- **TimeoutPeriod:** Start value of the timeout down-counter. This parameter must be a number between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_FDCAN_EnableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimeoutCounter

Function name

uint16_t HAL_FDCAN_GetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timeout counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Timeout:** counter value

HAL_FDCAN_ResetTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timeout counter to its start value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan, uint32_t TdcOffset, uint32_t TdcFilter)

Function description

Configure the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TdcOffset:** Transmitter Delay Compensation Offset. This parameter must be a number between 0x00 and 0x7F.
- **TdcFilter:** Transmitter Delay Compensation Filter Window Length. This parameter must be a number between 0x00 and 0x7F.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Start

Function name

HAL_StatusTypeDef HAL_FDCAN_Start (FDCAN_HandleTypeDef * hfdcan)

Function description

Start the FDCAN module.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Stop

Function name

HAL_StatusTypeDef HAL_FDCAN_Stop (FDCAN_HandleTypeDef * hfdcan)

Function description

Stop the FDCAN module and enable access to configuration registers.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxFifoQ

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxFifoQ (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxHeaderTypeDef * pTxHeader, uint8_t * pTxData)

Function description

Add a message to the Tx FIFO/Queue and activate the corresponding transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.

Return values

- **HAL:** status

HAL_FDCAN_GetLatestTxFifoQRequestBuffer

Function name

uint32_t HAL_FDCAN_GetLatestTxFifoQRequestBuffer (FDCAN_HandleTypeDef * hfdcan)

Function description

Get Tx buffer index of latest Tx FIFO/Queue request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** buffer index of last Tx FIFO/Queue request
  - Any value of
    - FDCAN_Tx_location if Tx request has been submitted.
  - 0 if no Tx FIFO/Queue request have been submitted.

HAL_FDCAN_AbortTxRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_AbortTxRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

Abort transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_GetRxMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_GetRxMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t RxLocation, FDCAN_RxHeaderTypeDef * pRxHeader, uint8_t * pRxData)

Function description

Get an FDCAN frame from the Rx FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxLocation:** Location of the received message to be read. This parameter can be a value of
  – FDCAN_Rx_location.
- **pRxHeader:** pointer to a FDCAN_RxHeaderTypeDef structure.
- **pRxData:** pointer to a buffer where the payload of the Rx frame will be stored.

Return values

- **HAL:** status

HAL_FDCAN_GetTxEvent

Function name

HAL_StatusTypeDef HAL_FDCAN_GetTxEvent (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxEventFifoTypeDef * pTxEvent)

Function description

Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxEvent:** pointer to a FDCAN_TxEventFifoTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetHighPriorityMessageStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetHighPriorityMessageStatus (FDCAN_HandleTypeDef * hfdcan, FDCAN_HpMsgStatusTypeDef * HpMsgStatus)

Function description

Get high priority message status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **HpMsgStatus:** pointer to an FDCAN_HpMsgStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetProtocolStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetProtocolStatus (FDCAN_HandleTypeDef * hfdcan, FDCAN_ProtocolStatusTypeDef * ProtocolStatus)

Function description

Get protocol status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ProtocolStatus:** pointer to an FDCAN_ProtocolStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetErrorCounters

Function name

HAL_StatusTypeDef HAL_FDCAN_GetErrorCounters (FDCAN_HandleTypeDef * hfdcan, FDCAN_ErrorCountersTypeDef * ErrorCounters)

Function description

Get error counter values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorCounters:** pointer to an FDCAN_ErrorCountersTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_IsTxBufferMessagePending

Function name

uint32_t HAL_FDCAN_IsTxBufferMessagePending (FDCAN_HandleTypeDef * hfdcan, uint32_t TxBufferIndex)

Function description

Check if a transmission request is pending on the selected Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxBufferIndex:** Tx buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **Status:**
  - 0 : No pending transmission request on TxBufferIndex list
  - 1 : Pending transmission request on TxBufferIndex.

HAL_FDCAN_GetRxFifoFillLevel

Function name

uint32_t HAL_FDCAN_GetRxFifoFillLevel (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo)

Function description

Return Rx FIFO fill level.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1

Return values

- **Rx:** FIFO fill level.

HAL_FDCAN_GetTxFifoFreeLevel

Function name

uint32_t HAL_FDCAN_GetTxFifoFreeLevel (FDCAN_HandleTypeDef * hfdcan)

Function description

Return Tx FIFO free level: number of consecutive free Tx FIFO elements starting from Tx FIFO GetIndex.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** FIFO free level.

HAL_FDCAN_IsRestrictedOperationMode

Function name

uint32_t HAL_FDCAN_IsRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Check if the FDCAN peripheral entered Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Status:**
  - 0 : Normal FDCAN operation.
  - 1 : Restricted Operation Mode active.

HAL_FDCAN_ExitRestrictedOperationMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t ITList, uint32_t InterruptLine)

Function description

Assign interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ITList:** indicates which interrupts group will be assigned to the selected interrupt line. This parameter can be any combination of
  - FDCAN_Interrupts_Group.
- **InterruptLine:** Interrupt line. This parameter can be a value of
  - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveITs, uint32_t BufferIndexes)

Function description

Enable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveITs:** indicates which interrupts will be enabled. This parameter can be any combination of
  - FDCAN_Interrupts.
- **BufferIndexes:** Tx Buffer Indexes. This parameter can be any combination of
  - FDCAN_Tx_location. This parameter is ignored if ActiveITs does not include one of the following:
    - FDCAN_IT_TX_COMPLETE
    - FDCAN_IT_TX_ABORT_COMPLETE

Return values

- **HAL:** status

HAL_FDCAN_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveITs)

Function description

Disable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveITs:** indicates which interrupts will be disabled. This parameter can be any combination of
  - FDCAN_Interrupts.

Return values

- **HAL:** status

HAL_FDCAN_IRQHandler

Function name

void HAL_FDCAN_IRQHandler (FDCAN_HandleTypeDef * hfdcan)

Function description

Handles FDCAN interrupt request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TxEventFifoCallback

Function name

void HAL_FDCAN_TxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TxEventFifoITs)

Function description

Tx Event callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxEventFifoITs:** indicates which Tx Event FIFO interrupts are signalled. This parameter can be any combination of
  - FDCAN_Tx_Event_Fifo_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo0Callback

Function name

void HAL_FDCAN_RxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo0ITs)

Function description

Rx FIFO 0 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo0ITs:** indicates which Rx FIFO 0 interrupts are signalled. This parameter can be any combination of
  - FDCAN_Rx_Fifo0_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo1Callback

Function name

void HAL_FDCAN_RxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo1ITs)

Function description

Rx FIFO 1 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo1ITs:** indicates which Rx FIFO 1 interrupts are signalled. This parameter can be any combination of
  - FDCAN_Rx_Fifo1_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxFifoEmptyCallback

Function name

void HAL_FDCAN_TxFifoEmptyCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Tx FIFO Empty callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TxBufferCompleteCallback

Function name

void HAL_FDCAN_TxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Complete callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the transmitted buffers. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_TxBufferAbortCallback

Function name

void HAL_FDCAN_TxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Cancellation callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the aborted buffers. This parameter can be any combination of
  – FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_HighPriorityMessageCallback

Function name

void HAL_FDCAN_HighPriorityMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

High Priority Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimestampWraparoundCallback

Function name

void HAL_FDCAN_TimestampWraparoundCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timestamp Wraparound callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimeoutOccurredCallback

Function name

void HAL_FDCAN_TimeoutOccurredCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timeout Occurred callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorCallback

Function name

void HAL_FDCAN_ErrorCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Error callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorStatusCallback

Function name

void HAL_FDCAN_ErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ErrorStatusITs)

Function description

Error status callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorStatusITs:** indicates which Error Status interrupts are signaled. This parameter can be any combination of
  - FDCAN_Error_Status_Interrupts.

Return values

- **None:**

HAL_FDCAN_GetError

Function name

uint32_t HAL_FDCAN_GetError (FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN error code.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **FDCAN:** Error Code

HAL_FDCAN_GetState

Function name

HAL_FDCAN_StateTypeDef HAL_FDCAN_GetState (FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** state

21.3 FDCAN Firmware driver defines

The following section lists the various define and macros of the module.

21.3.1 FDCAN

FDCAN

FDCAN Bit Rate Switching

FDCAN_BRS_OFF

FDCAN frames transmitted/received without bit rate switching

FDCAN_BRS_ON

FDCAN frames transmitted/received with bit rate switching

FDCAN Clock Divider

FDCAN_CLOCK_DIV1

Divide kernel clock by 1

FDCAN_CLOCK_DIV2

Divide kernel clock by 2

FDCAN_CLOCK_DIV4

Divide kernel clock by 4

FDCAN_CLOCK_DIV6

Divide kernel clock by 6

FDCAN_CLOCK_DIV8

Divide kernel clock by 8

FDCAN_CLOCK_DIV10

Divide kernel clock by 10

FDCAN_CLOCK_DIV12

Divide kernel clock by 12

FDCAN_CLOCK_DIV14

Divide kernel clock by 14

FDCAN_CLOCK_DIV16

Divide kernel clock by 16

FDCAN_CLOCK_DIV18

Divide kernel clock by 18

FDCAN_CLOCK_DIV20

Divide kernel clock by 20

FDCAN_CLOCK_DIV22

Divide kernel clock by 22

FDCAN_CLOCK_DIV24

Divide kernel clock by 24

FDCAN_CLOCK_DIV26

Divide kernel clock by 26

FDCAN_CLOCK_DIV28

Divide kernel clock by 28

FDCAN_CLOCK_DIV30

Divide kernel clock by 30

FDCAN communication state

FDCAN_COM_STATE_SYNC

Node is synchronizing on CAN communication

FDCAN_COM_STATE_IDLE

Node is neither receiver nor transmitter

FDCAN_COM_STATE_RX

Node is operating as receiver

FDCAN_COM_STATE_TX

Node is operating as transmitter

FDCAN Counter Interrupts

FDCAN_IT_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_IT_TIMEOUT_OCCURRED

Timeout reached

FDCAN Data Length Code

FDCAN_DLC_BYTES_0

0 bytes data field

FDCAN_DLC_BYTES_1

1 bytes data field

FDCAN_DLC_BYTES_2

2 bytes data field

FDCAN_DLC_BYTES_3

3 bytes data field

FDCAN_DLC_BYTES_4

4 bytes data field

FDCAN_DLC_BYTES_5

5 bytes data field

FDCAN_DLC_BYTES_6

6 bytes data field

FDCAN_DLC_BYTES_7

7 bytes data field

FDCAN_DLC_BYTES_8

8 bytes data field

FDCAN_DLC_BYTES_12

12 bytes data field

FDCAN_DLC_BYTES_16

16 bytes data field

FDCAN_DLC_BYTES_20

20 bytes data field

FDCAN_DLC_BYTES_24

24 bytes data field

FDCAN_DLC_BYTES_32

32 bytes data field

FDCAN_DLC_BYTES_48

48 bytes data field

FDCAN_DLC_BYTES_64

64 bytes data field

FDCAN Error Interrupts

FDCAN_IT_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_IT_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_IT_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_IT_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_IT_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_IT_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN Error State Indicator

FDCAN_ESI_ACTIVE

Transmitting node is error active

FDCAN_ESI_PASSIVE

Transmitting node is error passive

FDCAN Error Status Interrupts

FDCAN_IT_ERROR_PASSIVE

Error_Passive status changed

FDCAN_IT_ERROR_WARNING

Error_Warning status changed

FDCAN_IT_BUS_OFF

Bus_Off status changed

FDCAN Event Type

FDCAN_TX_EVENT

Tx event

FDCAN_TX_IN_SPITE_OF_ABORT

Transmission in spite of cancellation

FDCAN Exported Macros

__HAL_FDCAN_RESET_HANDLE_STATE

Description:

- Reset FDCAN handle state.

Parameters:

- __HANDLE__: FDCAN handle.

Return value:

- None

__HAL_FDCAN_ENABLE_IT

Description:

- Enable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_DISABLE_IT

Description:

- Disable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_IT

Description:

- Check whether the specified FDCAN interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be one of
  - FDCAN_Interrupts

Return value:

- ITStatus

__HAL_FDCAN_CLEAR_IT

Description:

- Clear the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the interrupts to clear. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_FLAG

Description:

- Check whether the specified FDCAN flag is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN flag. This parameter can be one of
  - FDCAN_flags

Return value:

- FlagStatus

__HAL_FDCAN_CLEAR_FLAG

Description:

- Clear the specified FDCAN flags.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the flags to clear. This parameter can be any combination of
  - FDCAN_flags

Return value:

- None

__HAL_FDCAN_GET_IT_SOURCE

Description:

- Check if the specified FDCAN interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN interrupt source to check. This parameter can be a value of
  - FDCAN_Interrupts

Return value:

- ITStatus

FDCAN Filter Configuration

FDCAN_FILTER_DISABLE

Disable filter element

FDCAN_FILTER_TO_RXFIFO0

Store in Rx FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1

Store in Rx FIFO 1 if filter matches

FDCAN_FILTER_REJECT

Reject ID if filter matches

FDCAN_FILTER_HP

Set high priority if filter matches

FDCAN_FILTER_TO_RXFIFO0_HP

Set high priority and store in FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1_HP

Set high priority and store in FIFO 1 if filter matches

FDCAN Filter Type

FDCAN_FILTER_RANGE

Range filter from FilterID1 to FilterID2

FDCAN_FILTER_DUAL

Dual ID filter for FilterID1 or FilterID2

FDCAN_FILTER_MASK

Classic filter: FilterID1 = filter, FilterID2 = mask

FDCAN_FILTER_RANGE_NO_EIDM

Range filter from FilterID1 to FilterID2, EIDM mask not applied

FDCAN Flags

FDCAN_FLAG_TX_COMPLETE

Transmission Completed

FDCAN_FLAG_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_FLAG_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN_FLAG_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_FLAG_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_FLAG_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_FLAG_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN_FLAG_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_FLAG_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_FLAG_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN_FLAG_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_FLAG_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_FLAG_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN_FLAG_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_FLAG_ERROR_PASSIVE

Error_Passive status changed

FDCAN_FLAG_ERROR_WARNING

Error_Warning status changed

FDCAN_FLAG_BUS_OFF

Bus_Off status changed

FDCAN_FLAG_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_FLAG_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_FLAG_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_FLAG_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN_FLAG_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_FLAG_TIMEOUT_OCCURRED

Timeout reached

FDCAN format

FDCAN_CLASSIC_CAN

Frame transmitted/received in Classic CAN format

FDCAN_FD_CAN

Frame transmitted/received in FDCAN format

FDCAN Frame Format

FDCAN_FRAME_CLASSIC

Classic mode

FDCAN_FRAME_FD_NO_BRS

FD mode without BitRate Switching

FDCAN_FRAME_FD_BRS

FD mode with BitRate Switching

FDCAN Frame Type

FDCAN_DATA_FRAME

Data frame

FDCAN_REMOTE_FRAME

Remote frame

FDCAN High Priority Message Storage

FDCAN_HP_STORAGE_NO_FIFO

No FIFO selected

FDCAN_HP_STORAGE_MSG_LOST

FIFO message lost

FDCAN_HP_STORAGE_RXFIFO0

Message stored in FIFO 0

FDCAN_HP_STORAGE_RXFIFO1

Message stored in FIFO 1

FDCAN ID Type

FDCAN_STANDARD_ID

Standard ID element

FDCAN_EXTENDED_ID

Extended ID element

FDCAN Interrupts Group

FDCAN_IT_GROUP_RX_FIFO0

RX FIFO 0 Interrupts Group: RF0LL: Rx FIFO 0 Message Lost RF0FL: Rx FIFO 0 is Full RF0NL: Rx FIFO 0 Has New Message

FDCAN_IT_GROUP_RX_FIFO1

RX FIFO 1 Interrupts Group: RF1LL: Rx FIFO 1 Message Lost RF1FL: Rx FIFO 1 is Full RF1NL: Rx FIFO 1 Has New Message

FDCAN_IT_GROUP_SMSG

Status Message Interrupts Group: TCFL: Transmission Cancellation Finished TCL: Transmission Completed HPML: High Priority Message

FDCAN_IT_GROUP_TX_FIFO_ERROR

TX FIFO Error Interrupts Group: TEFLL: Tx Event FIFO Element Lost TEFFL: Tx Event FIFO Full TEFNL: Tx Event FIFO New Entry TFEL: Tx FIFO Empty Interrupt Line

FDCAN_IT_GROUP_MISC

Misc. Interrupts Group: TOOL: Timeout Occurred MRAFL: Message RAM Access Failure TSWL: Timestamp Wraparound

FDCAN_IT_GROUP_BIT_LINE_ERROR

Bit and Line Error Interrupts Group: EPL: Error Passive ELOL: Error Logging Overflow

FDCAN_IT_GROUP_PROTOCOL_ERROR

Protocol Error Group: ARAL: Access to Reserved Address Line PEDL: Protocol Error in Data Phase Line PEAL: Protocol Error in Arbitration Phase Line WDIL: Watchdog Interrupt Line BOL: Bus_Off Status EWL: Warning Status

FDCAN Interrupts List

FDCAN_IT_LIST_RX_FIFO0

RX FIFO 0 Interrupts List

FDCAN_IT_LIST_RX_FIFO1

RX FIFO 1 Interrupts List

FDCAN_IT_LIST_SMSG

Status Message Interrupts List

FDCAN_IT_LIST_TX_FIFO_ERROR

TX FIFO Error Interrupts List

FDCAN_IT_LIST_MISC

Misc. Interrupts List

FDCAN_IT_LIST_BIT_LINE_ERROR

Bit and Line Error Interrupts List

FDCAN_IT_LIST_PROTOCOL_ERROR

Protocol Error Interrupts List

FDCAN interrupt line

FDCAN_INTERRUPT_LINE0

Interrupt Line 0

FDCAN_INTERRUPT_LINE1

Interrupt Line 1

FDCAN non-matching frames

FDCAN_ACCEPT_IN_RX_FIFO0

Accept in Rx FIFO 0

FDCAN_ACCEPT_IN_RX_FIFO1

Accept in Rx FIFO 1

FDCAN_REJECT

Reject

FDCAN Operating Mode

FDCAN_MODE_NORMAL

Normal mode

FDCAN_MODE_RESTRICTED_OPERATION

Restricted Operation mode

FDCAN_MODE_BUS_MONITORING

Bus Monitoring mode

FDCAN_MODE_INTERNAL_LOOPBACK

Internal LoopBack mode

FDCAN_MODE_EXTERNAL_LOOPBACK

External LoopBack mode

FDCAN protocol error code

FDCAN_PROTOCOL_ERROR_NONE

No error occurred

FDCAN_PROTOCOL_ERROR_STUFF

Stuff error

FDCAN_PROTOCOL_ERROR_FORM

Form error

FDCAN_PROTOCOL_ERROR_ACK

Acknowledge error

FDCAN_PROTOCOL_ERROR_BIT1

Bit 1 (recessive) error

FDCAN_PROTOCOL_ERROR_BIT0

Bit 0 (dominant) error

FDCAN_PROTOCOL_ERROR_CRC

CRC check sum error

FDCAN_PROTOCOL_ERROR_NO_CHANGE

No change since last read

FDCAN reject remote frames

FDCAN_FILTER_REMOTE

Filter remote frames

FDCAN_REJECT_REMOTE

Reject all remote frames

FDCAN Rx FIFO 0 Interrupts

FDCAN_IT_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_IT_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_IT_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN Rx FIFO 1 Interrupts

FDCAN_IT_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_IT_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_IT_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN FIFO operation mode

FDCAN_RX_FIFO_BLOCKING

Rx FIFO blocking mode

FDCAN_RX_FIFO_OVERWRITE

Rx FIFO overwrite mode

FDCAN Rx Interrupts

FDCAN_IT_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN Rx Location

FDCAN_RX_FIFO0

Get received message from Rx FIFO 0

FDCAN_RX_FIFO1

Get received message from Rx FIFO 1

FDCAN timeout operation

FDCAN_TIMEOUT_CONTINUOUS

Timeout continuous operation

FDCAN_TIMEOUT_TX_EVENT_FIFO

Timeout controlled by Tx Event FIFO

FDCAN_TIMEOUT_RX_FIFO0

Timeout controlled by Rx FIFO 0

FDCAN_TIMEOUT_RX_FIFO1

Timeout controlled by Rx FIFO 1

FDCAN timestamp

FDCAN_TIMESTAMP_INTERNAL

Timestamp counter value incremented according to TCP

FDCAN_TIMESTAMP_EXTERNAL

External timestamp counter value used

FDCAN timestamp prescaler

FDCAN_TIMESTAMP_PRESC_1

Timestamp counter time unit in equal to CAN bit time

FDCAN_TIMESTAMP_PRESC_2

Timestamp counter time unit in equal to CAN bit time multipled by 2

FDCAN_TIMESTAMP_PRESC_3

Timestamp counter time unit in equal to CAN bit time multipled by 3

FDCAN_TIMESTAMP_PRESC_4

Timestamp counter time unit in equal to CAN bit time multipled by 4

FDCAN_TIMESTAMP_PRESC_5

Timestamp counter time unit in equal to CAN bit time multipled by 5

FDCAN_TIMESTAMP_PRESC_6

Timestamp counter time unit in equal to CAN bit time multipled by 6

FDCAN_TIMESTAMP_PRESC_7

Timestamp counter time unit in equal to CAN bit time multipled by 7

FDCAN_TIMESTAMP_PRESC_8

Timestamp counter time unit in equal to CAN bit time multipled by 8

FDCAN_TIMESTAMP_PRESC_9

Timestamp counter time unit in equal to CAN bit time multipled by 9

FDCAN_TIMESTAMP_PRESC_10

Timestamp counter time unit in equal to CAN bit time multipled by 10

FDCAN_TIMESTAMP_PRESC_11

Timestamp counter time unit in equal to CAN bit time multipled by 11

FDCAN_TIMESTAMP_PRESC_12

Timestamp counter time unit in equal to CAN bit time multipled by 12

FDCAN_TIMESTAMP_PRESC_13

Timestamp counter time unit in equal to CAN bit time multipled by 13

FDCAN_TIMESTAMP_PRESC_14

Timestamp counter time unit in equal to CAN bit time multipled by 14

FDCAN_TIMESTAMP_PRESC_15

Timestamp counter time unit in equal to CAN bit time multipled by 15

FDCAN_TIMESTAMP_PRESC_16

Timestamp counter time unit in equal to CAN bit time multipled by 16

FDCAN Tx FIFO/Queue Mode

FDCAN_TX_FIFO_OPERATION

FIFO mode

FDCAN_TX_QUEUE_OPERATION

Queue mode

FDCAN Tx Event FIFO Interrupts

FDCAN_IT_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_IT_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_IT_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN Tx Interrupts

FDCAN_IT_TX_COMPLETE

Transmission Completed

FDCAN_IT_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_IT_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN Tx Location

FDCAN_TX_BUFFER0

Add message to Tx Buffer 0

FDCAN_TX_BUFFER1

Add message to Tx Buffer 1

FDCAN_TX_BUFFER2

Add message to Tx Buffer 2

22 HAL FLASH Generic Driver

22.1 FLASH Firmware driver registers structures

22.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32g4xx_hal_flash.h

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t Page***
- ***uint32_t NbPages***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***

  Mass erase or page erase. This parameter can be a value of ***FLASH_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks***

  Select bank to erase. This parameter must be a value of ***FLASH_Banks*** (FLASH_BANK_BOTH should be used only for mass erase)
- ***uint32_t FLASH_EraseInitTypeDef::Page***

  Initial Flash page to erase when page erase is disabled. This parameter must be a value between 0 and (max number of pages in the bank - 1) (eg : 127 for 512KB dual bank)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***

  Number of pages to be erased. This parameter must be a value between 1 and (max number of pages in the bank - value of initial page)

22.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32g4xx_hal_flash.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPArea***
- ***uint32_t WRPStartOffset***
- ***uint32_t WRPEndOffset***
- ***uint32_t RDPLevel***
- ***uint32_t USERType***
- ***uint32_t USERConfig***
- ***uint32_t PCROPConfig***
- ***uint32_t PCROPStartAddr***
- ***uint32_t PCROPEndAddr***
- ***uint32_t BootEntryPoint***
- ***uint32_t SecBank***
- ***uint32_t SecSize***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***

  Option byte to be configured. This parameter can be a combination of the values of ***FLASH_OB_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPArea***

  Write protection area to be programmed (used for OPTIONBYTE_WRP). Only one WRP area could be programmed at the same time. This parameter can be value of ***FLASH_OB_WRP_Area***

- ***uint32_t FLASH_OBProgramInitTypeDef::WRPStartOffset***

  Write protection start offset (used for OPTIONBYTE_WRP). This parameter must be a value between 0 and (max number of pages in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPEndOffset***

  Write protection end offset (used for OPTIONBYTE_WRP). This parameter must be a value between WRPStartOffset and (max number of pages in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPLevel***

  Set the read protection level.. (used for OPTIONBYTE_RDP). This parameter can be a value of ***FLASH_OB_Read_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERType***

  User option byte(s) to be configured (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERConfig***

  Value of the user option byte (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_BOR_LEVEL***, ***FLASH_OB_USER_nRST_STOP***, ***FLASH_OB_USER_nRST_STANDBY***, ***FLASH_OB_USER_nRST_SHUTDOWN***, ***FLASH_OB_USER_IWDG_SW***, ***FLASH_OB_USER_IWDG_STOP***, ***FLASH_OB_USER_IWDG_STANDBY***, ***FLASH_OB_USER_WWDG_SW***, ***FLASH_OB_USER_BFB2*** (*), ***FLASH_OB_USER_nBOOT1***, ***FLASH_OB_USER_SRAM_PE***, ***FLASH_OB_USER_CCMSRAM_RST***

  Note:

  – (*) availability depends on devices
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPConfig***

  Configuration of the PCROP (used for OPTIONBYTE_PCROP). This parameter must be a combination of ***FLASH_Banks*** (except FLASH_BANK_BOTH) and ***FLASH_OB_PCROP_RDP***
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPStartAddr***

  PCROP Start address (used for OPTIONBYTE_PCROP). This parameter must be a value between begin and end of bank => Be careful of the bank swapping for the address
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPEndAddr***

  PCROP End address (used for OPTIONBYTE_PCROP). This parameter must be a value between PCROP Start address and end of bank
- ***uint32_t FLASH_OBProgramInitTypeDef::BootEntryPoint***

  Set the Boot Lock (used for OPTIONBYTE_BOOT_LOCK). This parameter can be a value of ***FLASH_OB_Boot_Lock***
- ***uint32_t FLASH_OBProgramInitTypeDef::SecBank***

  Bank of securable memory area to be programmed (used for OPTIONBYTE_SEC). Only one securable memory area could be programmed at the same time. This parameter can be one of the following values: FLASH_BANK_1: Securable memory area to be programmed in bank 1 FLASH_BANK_2: Securable memory area to be programmed in bank 2 (*)

  Note:

  – (*) availability depends on devices
- ***uint32_t FLASH_OBProgramInitTypeDef::SecSize***

  Size of securable memory area to be programmed (used for OPTIONBYTE_SEC), in number of pages. Securable memory area is starting from first page of the bank. Only one securable memory could be programmed at the same time. This parameter must be a value between 0 and (max number of pages in the bank - 1)

22.1.3 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32g4xx_hal_flash.h

Data Fields

- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t Address***

- ***__IO uint32_t Bank***
- ***__IO uint32_t Page***
- ***__IO uint32_t NbPagesToErase***
- ***__IO FLASH_CacheTypeDef CacheToReactivate***

Field Documentation

- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
- ***__IO uint32_t FLASH_ProcessTypeDef::ErrorCode***
- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
- ***__IO uint32_t FLASH_ProcessTypeDef::Bank***
- ***__IO uint32_t FLASH_ProcessTypeDef::Page***
- ***__IO uint32_t FLASH_ProcessTypeDef::NbPagesToErase***
- ***__IO FLASH_CacheTypeDef FLASH_ProcessTypeDef::CacheToReactivate***

22.2 FLASH Firmware driver API description

The following section lists the various functions of the FLASH library.

22.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface accelerates code execution with a system of instruction prefetch and cache lines.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- Prefetch on I-Code
- 32 cache lines of 4*64 or 2*128 bits on I-Code
- 8 cache lines of 4*64 or 2*128 bits on D-Code
- Error code correction (ECC) : Data in flash are 72-bits word (8 bits added per double word)

22.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32G4xx devices.

1. Flash Memory IO Programming functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Program functions: double word and fast program (full row programming)
   - There are two modes of programming :
     - Polling mode using HAL_FLASH_Program() function
     - Interrupt mode using HAL_FLASH_Program_IT() function
2. Interrupts and flags management functions:
   - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
   - Callback functions are called when the flash operations are finished : HAL_FLASH_EndOfOperationCallback() when everything is ok, otherwise HAL_FLASH_OperationErrorCallback()
   - Get error flag status by calling HAL_GetError()

3. Option bytes management functions:
   – Lock and Unlock the option bytes using HAL_FLASH_OB_Unlock() and HAL_FLASH_OB_Lock() functions
   – Launch the reload of the option bytes using HAL_FLASH_Launch() function. In this case, a reset is generated

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the Instruction cache and the Data cache
- Reset the Instruction cache and the Data cache
- Enable/Disable the Flash power-down during low-power run and sleep modes
- Enable/Disable the Flash interrupts
- Monitor the Flash flags status

22.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

This section contains the following APIs:

- ***HAL_FLASH_Program***
- ***HAL_FLASH_Program_IT***
- ***HAL_FLASH_IRQHandler***
- ***HAL_FLASH_EndOfOperationCallback***
- ***HAL_FLASH_OperationErrorCallback***

22.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- ***HAL_FLASH_Unlock***
- ***HAL_FLASH_Lock***
- ***HAL_FLASH_OB_Unlock***
- ***HAL_FLASH_OB_Lock***
- ***HAL_FLASH_OB_Launch***

22.2.5 Peripheral Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

This section contains the following APIs:

- ***HAL_FLASH_GetError***

22.2.6 Detailed description of functions

HAL_FLASH_Program

Function name

HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)

Function description

Program double word or fast program of a row at a specified address.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type.
- **Address:** specifies the address to be programmed.
- **Data:** specifies the data to be programmed. This parameter is the data for the double word program and the address where are stored the data for the row fast program.

Return values

- **HAL_Status:**

HAL_FLASH_Program_IT

Function name

HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)

Function description

Program double word or fast program of a row at a specified address with interrupt enabled.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type.
- **Address:** specifies the address to be programmed.
- **Data:** specifies the data to be programmed. This parameter is the data for the double word program and the address where are stored the data for the row fast program.

Return values

- **HAL_Status:**

HAL_FLASH_IRQHandler

Function name

void HAL_FLASH_IRQHandler (void)

Function description

Handle FLASH interrupt request.

Return values

- **None:**

HAL_FLASH_EndOfOperationCallback

Function name

void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)

Function description

FLASH end of operation interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure:
  - Mass Erase: Bank number which has been requested to erase
  - Page Erase: Page which has been erased (if 0xFFFFFFFF, it means that all the selected pages have been erased)
  - Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_OperationErrorCallback

Function name

void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)

Function description

FLASH operation error interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure:
  - Mass Erase: Bank number which has been requested to erase
  - Page Erase: Page number which returned an error
  - Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_Unlock (void)

Function description

Unlock the FLASH control register access.

Return values

- **HAL_Status:**

HAL_FLASH_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_Lock (void)

Function description

Lock the FLASH control register access.

Return values

- **HAL_Status:**

HAL_FLASH_OB_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)

Function description

Unlock the FLASH Option Bytes Registers access.

Return values

- **HAL_Status:**

HAL_FLASH_OB_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)

Function description

Lock the FLASH Option Bytes Registers access.

Return values

- **HAL_Status:**

HAL_FLASH_OB_Launch

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)

Function description

Launch the option byte loading.

Return values

- **HAL_Status:**

HAL_FLASH_GetError

Function name

uint32_t HAL_FLASH_GetError (void)

Function description

Get the specific FLASH error flag.

Return values

- **FLASH_ErrorCode.:** The returned value can be:
  - HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP)
  - HAL_FLASH_ERROR_PGS: FLASH Programming Sequence error flag
  - HAL_FLASH_ERROR_PGP: FLASH Programming Parallelism error flag
  - HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag
  - HAL_FLASH_ERROR_WRP: FLASH Write protected error flag
  - HAL_FLASH_ERROR_OPERATION: FLASH operation Error flag
  - HAL_FLASH_ERROR_NONE: No error set
  - HAL_FLASH_ERROR_OP: FLASH Operation error
  - HAL_FLASH_ERROR_PROG: FLASH Programming error
  - HAL_FLASH_ERROR_WRP: FLASH Write protection error
  - HAL_FLASH_ERROR_PGA: FLASH Programming alignment error
  - HAL_FLASH_ERROR_SIZ: FLASH Size error
  - HAL_FLASH_ERROR_PGS: FLASH Programming sequence error
  - HAL_FLASH_ERROR_MIS: FLASH Fast programming data miss error
  - HAL_FLASH_ERROR_FAST: FLASH Fast programming error
  - HAL_FLASH_ERROR_RD: FLASH PCROP read error
  - HAL_FLASH_ERROR_OPTV: FLASH Option validity error

FLASH_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout)

Function description

Wait for a FLASH operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout.

Return values

- **HAL_Status:**

22.3 FLASH Firmware driver defines

The following section lists the various define and macros of the module.

22.3.1 FLASH

FLASH

FLASH Banks

FLASH_BANK_1

Bank 1

FLASH_BANK_2

Bank 2

FLASH_BANK_BOTH

Bank1 and Bank2

FLASH Error

HAL_FLASH_ERROR_NONE

HAL_FLASH_ERROR_OP

HAL_FLASH_ERROR_PROG

HAL_FLASH_ERROR_WRP

HAL_FLASH_ERROR_PGA

HAL_FLASH_ERROR_SIZ

HAL_FLASH_ERROR_PGS

HAL_FLASH_ERROR_MIS

HAL_FLASH_ERROR_FAST

HAL_FLASH_ERROR_RD

HAL_FLASH_ERROR_OPTV

HAL_FLASH_ERROR_ECCC

HAL_FLASH_ERROR_ECCD

HAL_FLASH_ERROR_ECCC2

HAL_FLASH_ERROR_ECCD2

FLASH Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH Latency. This parameter can be one of the following values :
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states
  - FLASH_LATENCY_5: FLASH Five wait states
  - FLASH_LATENCY_6: FLASH Six wait states
  - FLASH_LATENCY_7: FLASH Seven wait states
  - FLASH_LATENCY_8: FLASH Eight wait states
  - FLASH_LATENCY_9: FLASH Nine wait states
  - FLASH_LATENCY_10: FLASH Ten wait state
  - FLASH_LATENCY_11: FLASH Eleven wait state
  - FLASH_LATENCY_12: FLASH Twelve wait states
  - FLASH_LATENCY_13: FLASH Thirteen wait states
  - FLASH_LATENCY_14: FLASH Fourteen wait states
  - FLASH_LATENCY_15: FLASH Fifteen wait states

Return value:

- None

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH_Latency.: This parameter can be one of the following values :
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states
  - FLASH_LATENCY_5: FLASH Five wait states
  - FLASH_LATENCY_6: FLASH Six wait states
  - FLASH_LATENCY_7: FLASH Seven wait states
  - FLASH_LATENCY_8: FLASH Eight wait states
  - FLASH_LATENCY_9: FLASH Nine wait states
  - FLASH_LATENCY_10: FLASH Ten wait state
  - FLASH_LATENCY_11: FLASH Eleven wait state
  - FLASH_LATENCY_12: FLASH Twelve wait states
  - FLASH_LATENCY_13: FLASH Thirteen wait states
  - FLASH_LATENCY_14: FLASH Fourteen wait states
  - FLASH_LATENCY_15: FLASH Fifteen wait states

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:

- Enable the FLASH prefetch buffer.

Return value:

- None

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

Description:

- Disable the FLASH prefetch buffer.

Return value:

- None

__HAL_FLASH_INSTRUCTION_CACHE_ENABLE

Description:

- Enable the FLASH instruction cache.

Return value:

- none

__HAL_FLASH_INSTRUCTION_CACHE_DISABLE

Description:

- Disable the FLASH instruction cache.

Return value:

- none

__HAL_FLASH_DATA_CACHE_ENABLE

Description:

- Enable the FLASH data cache.

Return value:

- none

__HAL_FLASH_DATA_CACHE_DISABLE

Description:

- Disable the FLASH data cache.

Return value:

- none

__HAL_FLASH_INSTRUCTION_CACHE_RESET

Description:

- Reset the FLASH instruction Cache.

Return value:

- None

Notes:

- This function must be used only when the Instruction Cache is disabled.

__HAL_FLASH_DATA_CACHE_RESET

Description:

- Reset the FLASH data Cache.

Return value:

- None

Notes:

- This function must be used only when the data Cache is disabled.

__HAL_FLASH_POWER_DOWN_ENABLE

Notes:

- Writing this bit to 1, automatically the keys are lost and a new unlock sequence is necessary to re-write it to 0.

__HAL_FLASH_POWER_DOWN_DISABLE

Notes:

- Writing this bit to 0, automatically the keys are lost and a new unlock sequence is necessary to re-write it to 1.

__HAL_FLASH_SLEEP_POWERDOWN_ENABLE

Description:

- Enable the FLASH power down during Low-Power sleep mode.

Return value:

- none

__HAL_FLASH_SLEEP_POWERDOWN_DISABLE

Description:

- Disable the FLASH power down during Low-Power sleep mode.

Return value:

- none

FLASH Flags Definition

FLASH_FLAG_EOP

FLASH End of operation flag

FLASH_FLAG_OPERR

FLASH Operation error flag

FLASH_FLAG_PROGERR

FLASH Programming error flag

FLASH_FLAG_WRPERR

FLASH Write protection error flag

FLASH_FLAG_PGAERR

FLASH Programming alignment error flag

FLASH_FLAG_SIZERR

FLASH Size error flag

FLASH_FLAG_PGSERR

FLASH Programming sequence error flag

FLASH_FLAG_MISERR

FLASH Fast programming data miss error flag

FLASH_FLAG_FASTERR

FLASH Fast programming error flag

FLASH_FLAG_RDERR

FLASH PCROP read error flag

FLASH_FLAG_OPTVERR

FLASH Option validity error flag

FLASH_FLAG_BSY

FLASH Busy flag

FLASH_FLAG_ECCC

FLASH ECC correction in 64 LSB bits

FLASH_FLAG_ECCD

FLASH ECC detection in 64 LSB bits

FLASH_FLAG_ECCC2

FLASH ECC correction in 64 MSB bits (mode 128 bits only)

FLASH_FLAG_ECCD2

FLASH ECC detection in 64 MSB bits (mode 128 bits only)

FLASH_FLAG_SR_ERRORS

FLASH_FLAG_ECCR_ERRORS

FLASH_FLAG_ALL_ERRORS

FLASH Interrupts Macros

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP: End of FLASH Operation Interrupt
  - FLASH_IT_OPERR: Error Interrupt
  - FLASH_IT_RDERR: PCROP Read Error Interrupt
  - FLASH_IT_ECCC: ECC Correction Interrupt

Return value:

- none

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP: End of FLASH Operation Interrupt
  - FLASH_IT_OPERR: Error Interrupt
  - FLASH_IT_RDERR: PCROP Read Error Interrupt
  - FLASH_IT_ECCC: ECC Correction Interrupt

Return value:

- none

__HAL_FLASH_GET_FLAG

Description:

- Check whether the specified FLASH flag is set or not.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values:
  - FLASH_FLAG_EOP: FLASH End of Operation flag
  - FLASH_FLAG_OPERR: FLASH Operation error flag
  - FLASH_FLAG_PROGERR: FLASH Programming error flag
  - FLASH_FLAG_WRPERR: FLASH Write protection error flag
  - FLASH_FLAG_PGAERR: FLASH Programming alignment error flag
  - FLASH_FLAG_SIZERR: FLASH Size error flag
  - FLASH_FLAG_PGSERR: FLASH Programming sequence error flag
  - FLASH_FLAG_MISERR: FLASH Fast programming data miss error flag
  - FLASH_FLAG_FASTERR: FLASH Fast programming error flag
  - FLASH_FLAG_RDERR: FLASH PCROP read error flag
  - FLASH_FLAG_OPTVERR: FLASH Option validity error flag
  - FLASH_FLAG_BSY: FLASH write/erase operations in progress flag
  - FLASH_FLAG_ECCC: FLASH one ECC error has been detected and corrected in 64 LSB bits
  - FLASH_FLAG_ECCD: FLASH two ECC errors have been detected in 64 LSB bits
  - FLASH_FLAG_ECCC2(*): FLASH one ECC error has been detected and corrected in 64 MSB bits (mode 128 bits only)
  - FLASH_FLAG_ECCD2(*): FLASH two ECC errors have been detected in 64 MSB bits (mode 128 bits only)

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

Notes:

- (*) availability depends on devices

__HAL_FLASH_CLEAR_FLAG

Description:

- Clear the FLASH's pending flags.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  - FLASH_FLAG_EOP: FLASH End of Operation flag
  - FLASH_FLAG_OPERR: FLASH Operation error flag
  - FLASH_FLAG_PROGERR: FLASH Programming error flag
  - FLASH_FLAG_WRPERR: FLASH Write protection error flag
  - FLASH_FLAG_PGAERR: FLASH Programming alignment error flag
  - FLASH_FLAG_SIZERR: FLASH Size error flag
  - FLASH_FLAG_PGSERR: FLASH Programming sequence error flag
  - FLASH_FLAG_MISERR: FLASH Fast programming data miss error flag
  - FLASH_FLAG_FASTERR: FLASH Fast programming error flag
  - FLASH_FLAG_RDERR: FLASH PCROP read error flag
  - FLASH_FLAG_OPTVERR: FLASH Option validity error flag
  - FLASH_FLAG_ECCC: FLASH one ECC error has been detected and corrected in 64 LSB bits
  - FLASH_FLAG_ECCD: FLASH two ECC errors have been detected in 64 LSB bits
  - FLASH_FLAG_ECCC2(*): FLASH one ECC error has been detected and corrected in 64 MSB bits (mode 128 bits only)
  - FLASH_FLAG_ECCD2(*): FLASH two ECC errors have been detected in 64 MSB bits (mode 128 bits only)
  - FLASH_FLAG_SR_ERRORS: FLASH All SR errors flags
  - FLASH_FLAG_ECCR_ERRORS: FLASH All ECCR errors flags

Return value:

- None

Notes:

- (*) availability depends on devices

FLASH Interrupts Definition

FLASH_IT_EOP

End of FLASH Operation Interrupt source

FLASH_IT_OPERR

Error Interrupt source

FLASH_IT_RDERR

PCROP Read Error Interrupt source

FLASH_IT_ECCC

ECC Correction Interrupt source

FLASH Keys

FLASH_KEY1

Flash key1

FLASH_KEY2

Flash key2: used with FLASH_KEY1 to unlock the FLASH registers access

FLASH_PDKEY1

Flash power down key1

FLASH_PDKEY2

Flash power down key2: used with FLASH_PDKEY1 to unlock the RUN_PD bit in FLASH_ACR

FLASH_OPTKEY1

Flash option byte key1

FLASH_OPTKEY2

Flash option byte key2: used with FLASH_OPTKEY1 to allow option bytes operations

FLASH Latency

FLASH_LATENCY_0

FLASH Zero wait state

FLASH_LATENCY_1

FLASH One wait state

FLASH_LATENCY_2

FLASH Two wait states

FLASH_LATENCY_3

FLASH Three wait states

FLASH_LATENCY_4

FLASH Four wait states

FLASH_LATENCY_5

FLASH Five wait state

FLASH_LATENCY_6

FLASH Six wait state

FLASH_LATENCY_7

FLASH Seven wait states

FLASH_LATENCY_8

FLASH Eight wait states

FLASH_LATENCY_9

FLASH Nine wait states

FLASH_LATENCY_10

FLASH Ten wait state

FLASH_LATENCY_11

FLASH Eleven wait state

FLASH_LATENCY_12

FLASH Twelve wait states

FLASH_LATENCY_13

FLASH Thirteen wait states

FLASH_LATENCY_14

FLASH Fourteen wait states

FLASH_LATENCY_15

FLASH Fifteen wait states

FLASH Boot Lock

OB_BOOT_LOCK_DISABLE

Boot Lock Disable

OB_BOOT_LOCK_ENABLE

Boot Lock Enable

FLASH Option Bytes PCROP On RDP Level Type

OB_PCROP_RDP_NOT_ERASE

PCROP area is not erased when the RDP level is decreased from Level 1 to Level 0

OB_PCROP_RDP_ERASE

PCROP area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase)

FLASH Option Bytes Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2

Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0

FLASH Option Bytes Type

OPTIONBYTE_WRP

WRP option byte configuration

OPTIONBYTE_RDP

RDP option byte configuration

OPTIONBYTE_USER

USER option byte configuration

OPTIONBYTE_PCROP

PCROP option byte configuration

OPTIONBYTE_BOOT_LOCK

Boot lock option byte configuration

OPTIONBYTE_SEC

Securable memory option byte configuration

FLASH Option Bytes User BFB2 Mode

OB_BFB2_DISABLE

Dual-bank boot disable

OB_BFB2_ENABLE

Dual-bank boot enable

FLASH Option Bytes User BOR Level

OB_BOR_LEVEL_0

Reset level threshold is around 1.7V

OB_BOR_LEVEL_1

Reset level threshold is around 2.0V

OB_BOR_LEVEL_2

Reset level threshold is around 2.2V

OB_BOR_LEVEL_3

Reset level threshold is around 2.5V

OB_BOR_LEVEL_4

Reset level threshold is around 2.8V

FLASH Option Bytes User CCMSRAM Erase On Reset Type

OB_CCMSRAM_RST_ERASE

CCMSRAM erased when a system reset occurs

OB_CCMSRAM_RST_NOT_ERASE

CCMSRAM is not erased when a system reset occurs

FLASH Option Bytes User DBANK Type

OB_DBANK_128_BITS

Single-bank with 128-bits data

OB_DBANK_64_BITS

Dual-bank with 64-bits data

FLASH Option Bytes User internal reset holder bit

OB_IRH_DISABLE

Internal Reset holder disable

OB_IRH_ENABLE

Internal Reset holder enable

FLASH Option Bytes User IWDG Mode On Standby

OB_IWDG_STDBY_FREEZE

Independent watchdog counter is frozen in Standby mode

OB_IWDG_STDBY_RUN

Independent watchdog counter is running in Standby mode

FLASH Option Bytes User IWDG Mode On Stop

OB_IWDG_STOP_FREEZE

Independent watchdog counter is frozen in Stop mode

OB_IWDG_STOP_RUN

Independent watchdog counter is running in Stop mode

FLASH Option Bytes User IWDG Type

OB_IWDG_HW

Hardware independent watchdog

OB_IWDG_SW

Software independent watchdog

FLASH Option Bytes User nBOOT0 option bit

OB_nBOOT0_RESET

nBOOT0 = 0

OB_nBOOT0_SET

nBOOT0 = 1

FLASH Option Bytes User BOOT1 Type

OB_BOOT1_SRAM

Embedded SRAM1 is selected as boot space (if BOOT0=1)

OB_BOOT1_SYSTEM

System memory is selected as boot space (if BOOT0=1)

FLASH Option Bytes User NRST mode bit

OB_NRST_MODE_INPUT_ONLY

Reset pin is in Reset input mode only

OB_NRST_MODE_GPIO

Reset pin is in GPIO mode only

OB_NRST_MODE_INPUT_OUTPUT

Reset pin is in reset input and output mode

FLASH Option Bytes User Reset On Shutdown

OB_SHUTDOWN_RST

Reset generated when entering the shutdown mode

OB_SHUTDOWN_NORST

No reset generated when entering the shutdown mode

FLASH Option Bytes User Reset On Standby

OB_STANDBY_RST

Reset generated when entering the standby mode

OB_STANDBY_NORST

No reset generated when entering the standby mode

FLASH Option Bytes User Reset On Stop

OB_STOP_RST

Reset generated when entering the stop mode

OB_STOP_NORST

No reset generated when entering the stop mode

FLASH Option Bytes User Software BOOT0

OB_BOOT0_FROM_OB

BOOT0 taken from the option bit nBOOT0

OB_BOOT0_FROM_PIN

BOOT0 taken from PB8/BOOT0 pin

FLASH Option Bytes User SRAM Parity Check Type

OB_SRAM_PARITY_ENABLE

SRAM parity check enable (first 32kB of SRAM1 + CCM SRAM)

OB_SRAM_PARITY_DISABLE

SRAM parity check disable (first 32kB of SRAM1 + CCM SRAM)

FLASH Option Bytes User Type

OB_USER_BOR_LEV

BOR reset Level

OB_USER_nRST_STOP

Reset generated when entering the stop mode

OB_USER_nRST_STDBY

Reset generated when entering the standby mode

OB_USER_IWDG_SW

Independent watchdog selection

OB_USER_IWDG_STOP

Independent watchdog counter freeze in stop mode

OB_USER_IWDG_STDBY

Independent watchdog counter freeze in standby mode

OB_USER_WWDG_SW

Window watchdog selection

OB_USER_BFB2

Dual-bank boot

OB_USER_DBANK

Single bank with 128-bits data or two banks with 64-bits data

OB_USER_nBOOT1

Boot configuration

OB_USER_SRAM_PE

SRAM parity check enable (first 32kB of SRAM1 + CCM SRAM)

OB_USER_CCMSRAM_RST

CCMSRAM Erase when system reset

OB_USER_nRST_SHDW

Reset generated when entering the shutdown mode

OB_USER_nSWBOOT0

Software BOOT0

OB_USER_nBOOT0

nBOOT0 option bit

OB_USER_NRST_MODE

Reset pin configuration

OB_USER_IRHEN

Internal Reset Holder enable

FLASH Option Bytes User WWDG Type

OB_WWDG_HW

Hardware window watchdog

OB_WWDG_SW

Software window watchdog

FLASH WRP Area

OB_WRPAREA_BANK1_AREAA

Flash Bank 1 Area A

OB_WRPAREA_BANK1_AREAB

Flash Bank 1 Area B

OB_WRPAREA_BANK2_AREAA

Flash Bank 2 Area A

OB_WRPAREA_BANK2_AREAB

Flash Bank 2 Area B

FLASH Erase Type

FLASH_TYPEERASE_PAGES

Pages erase only

FLASH_TYPEERASE_MASSERASE

Flash mass erase activation

FLASH Program Type

FLASH_TYPEPROGRAM_DOUBLEWORD

Program a double-word (64-bit) at a specified address.

FLASH_TYPEPROGRAM_FAST

Fast program a 32 row double-word (64-bit) at a specified address. And another 32 row double-word (64-bit) will be programmed

FLASH_TYPEPROGRAM_FAST_AND_LAST

Fast program a 32 row double-word (64-bit) at a specified address. And this is the last 32 row double-word (64-bit) programmed

23 HAL FLASH Extension Driver

23.1 FLASHEx Firmware driver API description

The following section lists the various functions of the FLASHEx library.

23.1.1 Flash Extended features

Comparing to other previous devices, the FLASH interface for STM32G4xx devices contains the following additional features

- Capacity up to 512 Kbytes with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank 64-bits memory organization with possibility of single bank 128-bits
- Protected areas including WRP, PCROP and Securable memory

23.1.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32G4xx devices. It includes

1. Flash Memory Erase functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Erase function: Erase pages, or mass erase banks
   - There are two modes of erase :
     - Polling Mode using HAL_FLASHEx_Erase()
     - Interrupt Mode using HAL_FLASHEx_Erase_IT()
2. Option Bytes Programming function: Use HAL_FLASHEx_OBProgram() to:
   - Configure the write protection areas (WRP)
   - Set the Read protection Level (RDP)
   - Program the user Option Bytes
   - Configure the Proprietary Code ReadOut protection areas (PCROP)
   - Configure the Securable memory areas
   - Configure the Boot Lock
3. Get Option Bytes Configuration function: Use HAL_FLASHEx_OBGetConfig() to:
   - Get the configuration of write protection areas (WRP)
   - Get the level of read protection (RDP)
   - Get the value of the user Option Bytes
   - Get the configuration of Proprietary Code ReadOut Protection areas (PCROP)
   - Get the configuration of Securable memory areas
   - Get the status of Boot Lock
4. Activation of Securable memory area: Use HAL_FLASHEx_EnableSecMemProtection()
   - Deny the access to securable memory area
5. Enable or disable debugger: Use HAL_FLASHEx_EnableDebugger() or HAL_FLASHEx_DisableDebugger()

23.1.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extended FLASH programming operations Operations.

This section contains the following APIs:

- ***HAL_FLASHEx_Erase***
- ***HAL_FLASHEx_Erase_IT***
- ***HAL_FLASHEx_OBProgram***
- ***HAL_FLASHEx_OBGetConfig***
- ***HAL_FLASHEx_EnableSecMemProtection***

- *HAL_FLASHEx_EnableDebugger*
- *HAL_FLASHEx_DisableDebugger*

23.1.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)

Function description

Perform a mass erase or erase the specified FLASH memory pages.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **PageError:** pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased).

Return values

- **HAL_Status:**

HAL_FLASHEx_Erase_IT

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)

Function description

Perform a mass erase or erase the specified FLASH memory pages with interrupt enabled.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.

Return values

- **HAL_Status:**

HAL_FLASHEx_OBProgram

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Program Option bytes.

Parameters

- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.

Return values

- **HAL_Status:**

Notes

- To configure any option bytes, the option lock bit OPTLOCK must be cleared with the call of HAL_FLASH_OB_Unlock() function.
- New option bytes configuration will be taken into account in two cases: after an option bytes launch through the call of HAL_FLASH_OB_Launch()after a power reset (BOR reset or exit from Standby/Shutdown modes)

HAL_FLASHEx_OBGetConfig

Function name

void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Get the Option bytes configuration.

Parameters

- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information.

Return values

- **None:**

Notes

- The fields pOBInit->WRPArea and pOBInit->PCROPConfig should indicate which area is requested for the WRP and PCROP, else no information will be returned.

HAL_FLASHEx_EnableSecMemProtection

Function name

HAL_StatusTypeDef HAL_FLASHEx_EnableSecMemProtection (uint32_t Bank)

Function description

Enable the FLASH Securable Memory protection.

Parameters

- **Bank:** Bank to be protected This parameter can be one of the following values:
  - FLASH_BANK_1: Bank1 to be protected
  - FLASH_BANK_2: Bank2 to be protected (*)
  - FLASH_BANK_BOTH: Bank1 and Bank2 to be protected (*)

Return values

- **HAL:** Status

Notes

- (*) availability depends on devices

HAL_FLASHEx_EnableDebugger

Function name

void HAL_FLASHEx_EnableDebugger (void)

Function description

Enable Debugger.

Return values

- **None:**

Notes

- After calling this API, flash interface allow debugger intrusion.

HAL_FLASHEx_DisableDebugger

Function name

void HAL_FLASHEx_DisableDebugger (void)

Function description

Disable Debugger.

Return values

- **None:**

Notes

- After calling this API, Debugger is disabled: it's no more possible to break, see CPU register, etc...

FLASH_PageErase

Function name

void FLASH_PageErase (uint32_t Page, uint32_t Banks)

Function description

Erase the specified FLASH memory page.

Parameters

- **Page:** FLASH page to erase. This parameter must be a value between 0 and (max number of pages in the bank - 1).
- **Banks:** Bank where the page will be erased. This parameter can be one of the following values:
  - FLASH_BANK_1: Page in bank 1 to be erased
  - FLASH_BANK_2: Page in bank 2 to be erased (*)

Return values

- **None:**

Notes

- (*) availability depends on devices

FLASH_FlushCaches

Function name

void FLASH_FlushCaches (void)

Function description

Flush the instruction and data caches.

Return values

- **None:**

24 HAL FLASH__RAMFUNC Generic Driver

24.1 FLASH__RAMFUNC Firmware driver API description

The following section lists the various functions of the FLASH__RAMFUNC library.

24.1.1 Flash RAM functions

ARM Compiler

RAM functions are defined using the toolchain options. Functions that are executed in RAM should reside in a separate source module. Using the 'Options for File' dialog you can simply change the 'Code / Const' area of a module to a memory space in physical RAM. Available memory areas are declared in the 'Target' tab of the Options for Target' dialog.

ICCARM Compiler

RAM functions are defined using a specific toolchain keyword "__ramfunc".

GNU Compiler

RAM functions are defined using a specific toolchain attribute "__attribute__((section(".RamFunc")))".

24.1.2 ramfunc functions

This subsection provides a set of functions that should be executed from RAM.

This section contains the following APIs:

- *HAL_FLASHEx_EnableRunPowerDown*
- *HAL_FLASHEx_DisableRunPowerDown*
- *HAL_FLASHEx_OB_DBankConfig*

24.1.3 Detailed description of functions

HAL_FLASHEx_EnableRunPowerDown

Function name

__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_EnableRunPowerDown (void)

Function description

Enable the Power down in Run Mode.

Return values

- **None:**

Notes

- This function should be called and executed from SRAM memory.

HAL_FLASHEx_DisableRunPowerDown

Function name

__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_DisableRunPowerDown (void)

Function description

Disable the Power down in Run Mode.

Return values

- **None:**

Notes

- This function should be called and executed from SRAM memory.

HAL_FLASHEx_OB_DBankConfig

Function name

__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_OB_DBankConfig (uint32_t DBankConfig)

Function description

Program the FLASH DBANK User Option Byte.

Parameters

- **DBankConfig:** The FLASH DBANK User Option Byte value. This parameter can be one of the following values:
  - OB_DBANK_128_BITS: Single-bank with 128-bits data
  - OB_DBANK_64_BITS: Dual-bank with 64-bits data

Return values

- **HAL_Status:**

Notes

- To configure the user option bytes, the option lock bit OPTLOCK must be cleared with the call of the HAL_FLASH_OB_Unlock() function.
- To modify the DBANK option byte, no PCROP region should be defined. To deactivate PCROP, user should perform RDP changing.

25 HAL FMAC Generic Driver

25.1 FMAC Firmware driver registers structures

25.1.1 FMAC_HandleTypeDef

FMAC_HandleTypeDef is defined in the stm32g4xx_hal_fmac.h

Data Fields

- ***FMAC_TypeDef * Instance***
- ***uint32_t FilterParam***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***int16_t * pInput***
- ***uint16_t InputCurrentSize***
- ***uint16_t * pInputSize***
- ***int16_t * pOutput***
- ***uint16_t OutputCurrentSize***
- ***uint16_t * pOutputSize***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***DMA_HandleTypeDef * hdmaPreload***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_FMAC_StateTypeDef State***
- ***__IO HAL_FMAC_StateTypeDef RdState***
- ***__IO HAL_FMAC_StateTypeDef WrState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***FMAC_TypeDef* FMAC_HandleTypeDef::Instance***
  Register base address
- ***uint32_t FMAC_HandleTypeDef::FilterParam***
  Filter configuration (operation and parameters). Set to 0 if no valid configuration was applied.
- ***uint8_t FMAC_HandleTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint8_t FMAC_HandleTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***int16_t* FMAC_HandleTypeDef::pInput***
  Pointer to FMAC input data buffer
- ***uint16_t FMAC_HandleTypeDef::InputCurrentSize***
  Number of the input elements already written into FMAC
- ***uint16_t* FMAC_HandleTypeDef::pInputSize***
  Number of input elements to write (memory allocated to pInput). In case of early interruption of the filter operation, its value will be updated.
- ***int16_t* FMAC_HandleTypeDef::pOutput***
  Pointer to FMAC output data buffer
- ***uint16_t FMAC_HandleTypeDef::OutputCurrentSize***
  Number of the output elements already read from FMAC

- ***uint16_t* FMAC_HandleTypeDef::pOutputSize***
  Number of output elements to read (memory allocated to pOutput). In case of early interruption of the filter operation, its value will be updated.
- ***DMA_HandleTypeDef* FMAC_HandleTypeDef::hdmaIn***
  FMAC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* FMAC_HandleTypeDef::hdmaOut***
  FMAC peripheral output data DMA handle parameters
- ***DMA_HandleTypeDef* FMAC_HandleTypeDef::hdmaPreload***
  FMAC peripheral preloaded data (X1, X2 and Y) DMA handle parameters
- ***HAL_LockTypeDef FMAC_HandleTypeDef::Lock***
  FMAC locking object
- ***__IO HAL_FMAC_StateTypeDef FMAC_HandleTypeDef::State***
  FMAC state related to global handle management This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef FMAC_HandleTypeDef::RdState***
  FMAC state related to read operations (access to Y buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef FMAC_HandleTypeDef::WrState***
  FMAC state related to write operations (access to X1 buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO uint32_t FMAC_HandleTypeDef::ErrorCode***
  FMAC peripheral error code This parameter can be a value of ***FMAC_Error_Code***

25.1.2 FMAC_FilterConfigTypeDef

FMAC_FilterConfigTypeDef is defined in the stm32g4xx_hal_fmac.h

Data Fields

- ***uint8_t InputBaseAddress***
- ***uint8_t InputBufferSize***
- ***uint32_t InputThreshold***
- ***uint8_t CoeffBaseAddress***
- ***uint8_t CoeffBufferSize***
- ***uint8_t OutputBaseAddress***
- ***uint8_t OutputBufferSize***
- ***uint32_t OutputThreshold***
- ***int16_t * pCoeffA***
- ***uint8_t CoeffASize***
- ***int16_t * pCoeffB***
- ***uint8_t CoeffBSize***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***uint32_t Clip***
- ***uint32_t Filter***
- ***uint8_t P***
- ***uint8_t Q***
- ***uint8_t R***

Field Documentation

- ***uint8_t FMAC_FilterConfigTypeDef::InputBaseAddress***
  Base address of the input buffer (X1) within the internal memory (0x00 to 0xFF). Ignored if InputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.

- ***uint8_t FMAC_FilterConfigTypeDef::InputBufferSize***
  Number of 16-bit words allocated to the input buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::InputThreshold***
  Input threshold: the buffer full flag will be set if the number of free spaces in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBaseAddress***
  Base address of the coefficient buffer (X2) within the internal memory (0x00 to 0xFF). Ignored if CoeffBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBufferSize***
  Number of 16-bit words allocated to the coefficient buffer. 0 if a previous configuration should be kept.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBaseAddress***
  Base address of the output buffer (Y) within the internal memory (0x00 to 0xFF). Ignored if OuputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBufferSize***
  Number of 16-bit words allocated to the output buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::OutputThreshold***
  Output threshold: the buffer empty flag will be set if the number of unread values in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffA***
  [IIR only] Initialization of the coefficient vector A. If not needed, it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffASize***
  Size of the coefficient vector A.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffB***
  Initialization of the coefficient vector B. If not needed (re-use of a previously loaded buffer), it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBSize***
  Size of the coefficient vector B.
- ***uint8_t FMAC_FilterConfigTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint32_t FMAC_FilterConfigTypeDef::Clip***
  Enable or disable the clipping feature. If the q1.15 range is exceeded, wrapping is done when the clipping feature is disabled and saturation is done when the clipping feature is enabled. This parameter can be a value of ***FMAC_Clip_State***.
- ***uint32_t FMAC_FilterConfigTypeDef::Filter***
  Filter type. This parameter can be a value of ***FMAC_Functions*** (filter related values).
- ***uint8_t FMAC_FilterConfigTypeDef::P***
  Parameter P (vector length, number of filter taps, etc.).
- ***uint8_t FMAC_FilterConfigTypeDef::Q***
  Parameter Q (vector length, etc.). Ignored if not needed.
- ***uint8_t FMAC_FilterConfigTypeDef::R***
  Parameter R (gain, etc.). Ignored if not needed.

25.2 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

25.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the FMAC peripheral and the associated handle
- DeInitialize the FMAC peripheral
- Initialize the FMAC MSP (MCU Specific Package)
- De-Initialize the FMAC MSP
- Register a User FMAC Callback
- Unregister a FMAC CallBack

This section contains the following APIs:

- ***HAL_FMAC_Init***
- ***HAL_FMAC_DeInit***
- ***HAL_FMAC_MspInit***
- ***HAL_FMAC_MspDeInit***

25.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure the FMAC peripheral: memory area, filter type and parameters, way to access to the input and output memory area (none, polling, IT, DMA).
- Start the FMAC processing (filter).
- Handle the input data that will be provided into FMAC.
- Handle the output data provided by FMAC.
- Stop the FMAC processing (filter).

This section contains the following APIs:

- ***HAL_FMAC_FilterConfig***
- ***HAL_FMAC_FilterConfig_DMA***
- ***HAL_FMAC_FilterPreload***
- ***HAL_FMAC_FilterPreload_DMA***
- ***HAL_FMAC_FilterStart***
- ***HAL_FMAC_AppendFilterData***
- ***HAL_FMAC_ConfigFilterOutputBuffer***
- ***HAL_FMAC_PollFilterData***
- ***HAL_FMAC_FilterStop***

25.2.3 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt: the user's input data is half written (DMA only) or completely written.
- DMA or Interrupt: the user's output buffer is half filled (DMA only) or completely filled.
- DMA or Interrupt: error handling.

This section contains the following APIs:

- ***HAL_FMAC_ErrorCallback***
- ***HAL_FMAC_HalfGetDataCallback***
- ***HAL_FMAC_GetDataCallback***
- ***HAL_FMAC_HalfOutputDataReadyCallback***
- ***HAL_FMAC_OutputDataReadyCallback***
- ***HAL_FMAC_FilterConfigCallback***
- ***HAL_FMAC_FilterPreloadCallback***

25.2.4 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_FMAC_IRQHandler*

25.2.5 Peripheral State and Error functions

This subsection provides functions allowing to

- Check the FMAC state
- Get error code

This section contains the following APIs:

- *HAL_FMAC_GetState*
- *HAL_FMAC_GetError*

25.2.6 Detailed description of functions

HAL_FMAC_Init

Function name

HAL_StatusTypeDef HAL_FMAC_Init (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC peripheral and the associated handle.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_DeInit

Function name

HAL_StatusTypeDef HAL_FMAC_DeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC peripheral.

Parameters

- **hfmac:** pointer to a FMAC structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_MspInit

Function name

void HAL_FMAC_MspInit (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_MspDeInit

Function name

void HAL_FMAC_MspDeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_FilterConfig

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using polling mode.

HAL_FMAC_FilterConfig_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig_DMA (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using DMA.

HAL_FMAC_FilterPreload

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using polling mode.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterPreload_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload_DMA (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using DMA.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterStart

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStart (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Start the FMAC processing according to the existing FMAC configuration.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** pointer to buffer where output data of FMAC processing will be stored in the next steps. If it is set to NULL, the output will not be read and it will be up to an external IP to empty the output buffer.
- **pOutputSize:** pointer to the size of the output buffer. The number of read data will be written here.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_AppendFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_AppendFilterData (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint16_t * pInputSize)

Function description

Provide a new input buffer that will be loaded into the FMAC input memory area.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** New input vector (additional input data).
- **pInputSize:** Size of the input vector (if all the data can't be written, it will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ConfigFilterOutputBuffer

Function name

HAL_StatusTypeDef HAL_FMAC_ConfigFilterOutputBuffer (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Provide a new output buffer to be filled with the data computed by FMAC unit.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** New output vector.
- **pOutputSize:** Size of the output vector (if the vector can't be entirely filled, pOutputSize will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_PollFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_PollFilterData (FMAC_HandleTypeDef * hfmac, uint32_t Timeout)

Function description

Handle the input and/or output data in polling mode.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **Timeout:** timeout value.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- This function writes the previously provided user's input data and fills the previously provided user's output buffer, according to the existing FMAC configuration (polling mode only). The function returns when the input data has been handled or when the output data is filled. The possible unused data isn't kept. It will be up to the user to handle it. The previously provided pInputSize and pOutputSize will be used to indicate to the size of the read/written data to the user.

HAL_FMAC_FilterStop

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStop (FMAC_HandleTypeDef * hfmac)

Function description

Stop the FMAC processing.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ErrorCallback

Function name

void HAL_FMAC_ErrorCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC error callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfGetDataCallback

Function name

void HAL_FMAC_HalfGetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get half data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetDataCallback

Function name

void HAL_FMAC_GetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfOutputDataReadyCallback

Function name

void HAL_FMAC_HalfOutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC half output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_OutputDataReadyCallback

Function name

void HAL_FMAC_OutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterConfigCallback

Function name

void HAL_FMAC_FilterConfigCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter configuration callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterPreloadCallback

Function name

void HAL_FMAC_FilterPreloadCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter preload callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_IRQHandler

Function name

void HAL_FMAC_IRQHandler (FMAC_HandleTypeDef * hfmac)

Function description

Handle FMAC interrupt request.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetState

Function name

HAL_FMAC_StateTypeDef HAL_FMAC_GetState (FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC state.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_FMAC_StateTypeDef:** FMAC state

HAL_FMAC_GetError

Function name

uint32_t HAL_FMAC_GetError (FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC peripheral error.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **uint32_t:** Error bit-map based on FMAC Error code

Notes

- The returned error is a bit-map combination of possible errors.

25.3 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

25.3.1 FMAC

FMAC

FMAC Buffer Access

FMAC_BUFFER_ACCESS_NONE

Buffer handled by an external IP (ADC for instance)

FMAC_BUFFER_ACCESS_DMA

Buffer accessed through DMA

FMAC_BUFFER_ACCESS_POLLING

Buffer accessed through polling

FMAC_BUFFER_ACCESS_IT

Buffer accessed through interruptions

FMAC Clip State

FMAC_CLIP_DISABLED

Clipping disabled

FMAC_CLIP_ENABLED

Clipping enabled

FMAC Data Buffer Threshold

FMAC_THRESHOLD_1

Input: Buffer full flag set if the number of free spaces in the buffer is less than 1. Output: Buffer empty flag set if the number of unread values in the buffer is less than 1.

FMAC_THRESHOLD_2

Input: Buffer full flag set if the number of free spaces in the buffer is less than 2. Output: Buffer empty flag set if the number of unread values in the buffer is less than 2.

FMAC_THRESHOLD_4

Input: Buffer full flag set if the number of free spaces in the buffer is less than 4. Output: Buffer empty flag set if the number of unread values in the buffer is less than 4.

FMAC_THRESHOLD_8

Input: Buffer full flag set if the number of free spaces in the buffer is less than 8. Output: Buffer empty flag set if the number of unread values in the buffer is less than 8.

FMAC_THRESHOLD_NO_VALUE

The configured threshold value shouldn't be changed

FMAC Error code

HAL_FMAC_ERROR_NONE

No error

HAL_FMAC_ERROR_SAT

Saturation error

HAL_FMAC_ERROR_UNFL

Underflow error

HAL_FMAC_ERROR_OVFL

Overflow error

HAL_FMAC_ERROR_DMA

DMA error

HAL_FMAC_ERROR_RESET

Reset error

HAL_FMAC_ERROR_PARAM

Parameter error

HAL_FMAC_ERROR_TIMEOUT

Timeout error

FMAC Exported Macros

__HAL_FMAC_RESET_HANDLE_STATE

Description:

- Reset FMAC handle state.

Parameters:

- __HANDLE__: FMAC handle.

Return value:

- None

__HAL_FMAC_ENABLE_IT

Description:

- Enable the specified FMAC interrupt.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- None

__HAL_FMAC_DISABLE_IT

Description:

- Disable the FMAC interrupt.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- None

__HAL_FMAC_GET_IT

Description:

- Check whether the specified FMAC interrupt occurred or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be any combination of the following values:
  - FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  - FMAC_FLAG_X1FULL X1 Buffer Full Flag
  - FMAC_FLAG_OVFL Overflow Error Flag
  - FMAC_FLAG_UNFL Underflow Error Flag
  - FMAC_FLAG_SAT Saturation Error Flag

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_FMAC_CLEAR_IT

Description:

- Clear specified FMAC interrupt status.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to clear.

Return value:

- None

__HAL_FMAC_GET_FLAG

Description:

- Check whether the specified FMAC status flag is set or not.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to check. This parameter can be any combination of the following values:
  - FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  - FMAC_FLAG_X1FULL X1 Buffer Full Flag
  - FMAC_FLAG_OVFL Overflow Error Flag
  - FMAC_FLAG_UNFL Underflow Error Flag
  - FMAC_FLAG_SAT Saturation error Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_FMAC_CLEAR_FLAG

Description:

- Clear specified FMAC status flag.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to clear.

Return value:

- None

__HAL_FMAC_GET_IT_SOURCE

Description:

- Check whether the specified FMAC interrupt is enabled or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be one of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- FlagStatus

FMAC status flags

FMAC_FLAG_YEMPTY

Y Buffer Empty Flag

FMAC_FLAG_X1FULL

X1 Buffer Full Flag

FMAC_FLAG_OVFL

Overflow Error Flag

FMAC_FLAG_UNFL

Underflow Error Flag

FMAC_FLAG_SAT

Saturation Error Flag (this helps in debugging a filter)

FMAC Functions

FMAC_FUNC_LOAD_X1

Load X1 buffer

FMAC_FUNC_LOAD_X2

Load X2 buffer

FMAC_FUNC_LOAD_Y

Load Y buffer

FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

FMAC Interrupts Enable bit

FMAC_IT_RIEN

Read Interrupt Enable

FMAC_IT_WIEN

Write Interrupt Enable

FMAC_IT_OVFLIEN

Overflow Error Interrupt Enable

FMAC_IT_UNFLIEN

Underflow Error Interrupt Enable

FMAC_IT_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

26 HAL GPIO Generic Driver

26.1 GPIO Firmware driver registers structures

26.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32g4xx_hal_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_pins***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

26.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

26.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.
- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

- The external interrupt/event controller consists of up to 44 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

26.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To lock pin configuration until next reset use HAL_GPIO_LockPin().
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PF0 and PF1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

26.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_GPIO_Init***
- ***HAL_GPIO_DeInit***

26.2.4 Detailed description of functions

HAL_GPIO_Init

Function name

void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)

Function description

Initialize the GPIOx peripheral according to the specified parameters in the GPIO_Init.

Parameters

- **GPIOx:** where x can be (A..G) to select the GPIO peripheral for STM32G4xx family
- **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

HAL_GPIO_DeInit

Function name

void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)

Function description

De-initialize the GPIOx peripheral registers to their default reset values.

Parameters

- **GPIOx:** where x can be (A..G) to select the GPIO peripheral for STM32G4xx family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Read the specified input port pin.

Parameters

- **GPIOx:** where x can be (A..G) to select the GPIO peripheral for STM32G4xx family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Set or clear the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..G) to select the GPIO peripheral for STM32G4xx family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  - GPIO_PIN_RESET: to clear the port pin
  - GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_TogglePin

Function name

void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Toggle the specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..G) to select the GPIO peripheral for STM32G4xx family
- **GPIO_Pin:** specifies the pin to be toggled. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_LockPin

Function name

HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Lock GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..G) to select the GPIO peripheral for STM32G4xx family
- **GPIO_Pin:** specifies the port bits to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

Function description

Handle EXTI interrupt request.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Callback

Function name

void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)

Function description

EXTI line detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

26.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

26.3.1 GPIO

GPIO

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Check whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clear the EXTI's line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT

Description:

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- None

GPIO mode

GPIO_MODE_INPUT

Input Floating Mode

GPIO_MODE_OUTPUT_PP

Output Push Pull Mode

GPIO_MODE_OUTPUT_OD

Output Open Drain Mode

GPIO_MODE_AF_PP

Alternate Function Push Pull Mode

GPIO_MODE_AF_OD

Alternate Function Open Drain Mode

GPIO_MODE_ANALOG

Analog Mode

GPIO_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO pins

GPIO_PIN_0

GPIO_PIN_1

GPIO_PIN_2

GPIO_PIN_3

GPIO_PIN_4

GPIO_PIN_5

GPIO_PIN_6

GPIO_PIN_7

GPIO_PIN_8

GPIO_PIN_9

GPIO_PIN_10

GPIO_PIN_11

GPIO_PIN_12

GPIO_PIN_13

GPIO_PIN_14

GPIO_PIN_15

GPIO_PIN_All

GPIO_PIN_MASK

GPIO pull

GPIO_NOPULL

No Pull-up or Pull-down activation

GPIO_PULLUP

Pull-up activation

GPIO_PULLDOWN

Pull-down activation

GPIO speed

GPIO_SPEED_FREQ_LOW

range up to 5 MHz, please refer to the product datasheet

GPIO_SPEED_FREQ_MEDIUM

range 5 MHz to 25 MHz, please refer to the product datasheet

GPIO_SPEED_FREQ_HIGH

range 25 MHz to 50 MHz, please refer to the product datasheet

GPIO_SPEED_FREQ_VERY_HIGH

range 50 MHz to 120 MHz, please refer to the product datasheet

27 HAL GPIO Extension Driver

27.1 GPIOEx Firmware driver defines

The following section lists the various define and macros of the module.

27.1.1 GPIOEx

GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_RTC_50Hz

GPIO_AF0_MCO

GPIO_AF0_SWJ

GPIO_AF0_TRACE

GPIO_AF1_TIM2

GPIO_AF1_TIM5

GPIO_AF1_TIM16

GPIO_AF1_TIM17

GPIO_AF1_TIM17_COMP1

GPIO_AF1_TIM15

GPIO_AF1_LPTIM1

GPIO_AF1_IR

GPIO_AF2_TIM1

GPIO_AF2_TIM2

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM5

GPIO_AF2_TIM8

GPIO_AF2_TIM15

GPIO_AF2_TIM16

GPIO_AF2_TIM20

GPIO_AF2_TIM1_COMP1

GPIO_AF2_TIM15_COMP1

GPIO_AF2_TIM16_COMP1

GPIO_AF2_TIM20_COMP1

GPIO_AF2_TIM20_COMP2

GPIO_AF2_I2C3

GPIO_AF2_COMP1

GPIO_AF3_TIM15

GPIO_AF3_TIM20

GPIO_AF3_UCPD1

GPIO_AF3_I2C3

GPIO_AF3_I2C4

GPIO_AF3_HRTIM1

GPIO_AF3_QUADSPI

GPIO_AF3_TIM8

GPIO_AF3_SAI1

GPIO_AF3_COMP3

GPIO_AF4_TIM1

GPIO_AF4_TIM8

GPIO_AF4_TIM16

GPIO_AF4_TIM17

GPIO_AF4_TIM8_COMP1

GPIO_AF4_I2C1

GPIO_AF4_I2C2

GPIO_AF4_I2C3

GPIO_AF4_I2C4

GPIO_AF5_SPI1

GPIO_AF5_SPI2

GPIO_AF5_SPI4

GPIO_AF5_IR

GPIO_AF5_TIM8

GPIO_AF5_TIM8_COMP1

GPIO_AF5_UART4

GPIO_AF5_UART5

GPIO_AF5_I2S2ext

GPIO_AF6_SPI2

GPIO_AF6_SPI3

GPIO_AF6_TIM1

GPIO_AF6_TIM5

GPIO_AF6_TIM8

GPIO_AF6_TIM20

GPIO_AF6_TIM1_COMP1

GPIO_AF6_TIM1_COMP2

GPIO_AF6_TIM8_COMP2

GPIO_AF6_IR

GPIO_AF6_I2S3ext

GPIO_AF7_USART1

GPIO_AF7_USART2

GPIO_AF7_USART3

GPIO_AF7_COMP5

GPIO_AF7_COMP6

GPIO_AF7_COMP7

GPIO_AF8_COMP1

GPIO_AF8_COMP2

GPIO_AF8_COMP3

GPIO_AF8_COMP4

GPIO_AF8_COMP5

GPIO_AF8_COMP6

GPIO_AF8_COMP7

GPIO_AF8_I2C3

GPIO_AF8_I2C4

GPIO_AF8_LPUART1

GPIO_AF8_UART4

GPIO_AF8_UART5

GPIO_AF9_TIM1

GPIO_AF9_TIM8

GPIO_AF9_TIM15

GPIO_AF9_TIM1_COMP1

GPIO_AF9_TIM8_COMP1

GPIO_AF9_TIM15_COMP1

GPIO_AF9_FDCAN1

GPIO_AF9_FDCAN2

GPIO_AF10_TIM2

GPIO_AF10_TIM3

GPIO_AF10_TIM4

GPIO_AF10_TIM8

GPIO_AF10_TIM17

GPIO_AF10_TIM8_COMP2

GPIO_AF10_TIM17_COMP1

GPIO_AF10_QUADSPI

GPIO_AF11_FDCAN1

GPIO_AF11_FDCAN3

GPIO_AF11_TIM1

GPIO_AF11_TIM8

GPIO_AF11_TIM8_COMP1

GPIO_AF11_LPTIM1

GPIO_AF12_LPUART1

GPIO_AF12_TIM1

GPIO_AF12_TIM1_COMP1

GPIO_AF12_TIM1_COMP2

GPIO_AF12_HRTIM1

GPIO_AF12_FMC

GPIO_AF12_SAI1

GPIO_AF13_HRTIM1

GPIO_AF13_SAI1

GPIO_AF14_TIM2

GPIO_AF14_TIM15

GPIO_AF14_UCPD1

GPIO_AF14_SAI1

GPIO_AF14_UART4

GPIO_AF14_UART5

GPIO_AF15_EVENTOUT

IS_GPIO_AF

GPIOEx Get Port Index

GPIO_GET_INDEX

28 HAL HRTIM Generic Driver

28.1 HRTIM Firmware driver registers structures

28.1.1 HRTIM_InitTypeDef

HRTIM_InitTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t HRTIMInterruptResquests***
- ***uint32_t SyncOptions***
- ***uint32_t SyncInputSource***
- ***uint32_t SyncOutputSource***
- ***uint32_t SyncOutputPolarity***

Field Documentation

- ***uint32_t HRTIM_InitTypeDef::HRTIMInterruptResquests***

  Specifies which interrupts requests must enabled for the HRTIM instance. This parameter can be any combination of ***HRTIM_Common_Interrupt_Enable***
- ***uint32_t HRTIM_InitTypeDef::SyncOptions***

  Specifies how the HRTIM instance handles the external synchronization signals. The HRTIM instance can be configured to act as a slave (waiting for a trigger to be synchronized) or a master (generating a synchronization signal) or both. This parameter can be a combination of ***HRTIM_Synchronization_Options***.
- ***uint32_t HRTIM_InitTypeDef::SyncInputSource***

  Specifies the external synchronization input source (significant only when the HRTIM instance is configured as a slave). This parameter can be a value of ***HRTIM_Synchronization_Input_Source***.
- ***uint32_t HRTIM_InitTypeDef::SyncOutputSource***

  Specifies the source and event to be sent on the external synchronization outputs (significant only when the HRTIM instance is configured as a master). This parameter can be a value of ***HRTIM_Synchronization_Output_Source***
- ***uint32_t HRTIM_InitTypeDef::SyncOutputPolarity***

  Specifies the conditioning of the event to be sent on the external synchronization outputs (significant only when the HRTIM instance is configured as a master). This parameter can be a value of ***HRTIM_Synchronization_Output_Polarity***

28.1.2 HRTIM_TimerParamTypeDef

HRTIM_TimerParamTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t CaptureTrigger1***
- ***uint32_t CaptureTrigger2***
- ***uint32_t InterruptRequests***
- ***uint32_t DMARequests***
- ***uint32_t DMASrcAddress***
- ***uint32_t DMADstAddress***
- ***uint32_t DMASize***

Field Documentation

- ***uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger1***

  Event(s) triggering capture unit 1. When the timer operates in Simple mode, this parameter can be a value of ***HRTIM_External_Event_Channels***. When the timer operates in Waveform mode, this parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***.

- ***uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger2***
  Event(s) triggering capture unit 2. When the timer operates in Simple mode, this parameter can be a value of ***HRTIM_External_Event_Channels***. When the timer operates in Waveform mode, this parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***.
- ***uint32_t HRTIM_TimerParamTypeDef::InterruptRequests***
  Interrupts requests enabled for the timer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMARequests***
  DMA requests enabled for the timer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMASrcAddress***
  Address of the source address of the DMA transfer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMADstAddress***
  Address of the destination address of the DMA transfer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMASize***
  Size of the DMA transfer

28.1.3 HRTIM_HandleTypeDef

HRTIM_HandleTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***HRTIM_TypeDef * Instance***
- ***HRTIM_InitTypeDef Init***
- ***HRTIM_TimerParamTypeDef TimerParam***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HRTIM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdmaMaster***
- ***DMA_HandleTypeDef * hdmaTimerA***
- ***DMA_HandleTypeDef * hdmaTimerB***
- ***DMA_HandleTypeDef * hdmaTimerC***
- ***DMA_HandleTypeDef * hdmaTimerD***
- ***DMA_HandleTypeDef * hdmaTimerE***
- ***DMA_HandleTypeDef * hdmaTimerF***

Field Documentation

- ***HRTIM_TypeDef* HRTIM_HandleTypeDef::Instance***
  Register base address
- ***HRTIM_InitTypeDef HRTIM_HandleTypeDef::Init***
  HRTIM required parameters
- ***HRTIM_TimerParamTypeDef HRTIM_HandleTypeDef::TimerParam[MAX_HRTIM_TIMER]***
  HRTIM timers - including the master - parameters
- ***HAL_LockTypeDef HRTIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_HRTIM_StateTypeDef HRTIM_HandleTypeDef::State***
  HRTIM communication state
- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaMaster***
  Master timer DMA handle parameters
- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaTimerA***
  Timer A DMA handle parameters
- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaTimerB***
  Timer B DMA handle parameters
- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaTimerC***
  Timer C DMA handle parameters

- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaTimerD***
  Timer D DMA handle parameters
- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaTimerE***
  Timer E DMA handle parameters
- ***DMA_HandleTypeDef* HRTIM_HandleTypeDef::hdmaTimerF***
  Timer F DMA handle parameters

28.1.4 HRTIM_TimeBaseCfgTypeDef

HRTIM_TimeBaseCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Period***
- ***uint32_t RepetitionCounter***
- ***uint32_t PrescalerRatio***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t HRTIM_TimeBaseCfgTypeDef::Period***
  Specifies the timer period. The period value must be above 3 periods of the fHRTIM clock. Maximum value is = 0xFFDFU
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::RepetitionCounter***
  Specifies the timer repetition period. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::PrescalerRatio***
  Specifies the timer clock prescaler ratio. This parameter can be any value of ***HRTIM_Prescaler_Ratio***
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::Mode***
  Specifies the counter operating mode. This parameter can be any value of ***HRTIM_Counter_Operating_Mode***

28.1.5 HRTIM_SimpleOCChannelCfgTypeDef

HRTIM_SimpleOCChannelCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Pulse***
- ***uint32_t Polarity***
- ***uint32_t IdleLevel***

Field Documentation

- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Mode***
  Specifies the output compare mode (toggle, active, inactive). This parameter can be any value of of ***HRTIM_Simple_OC_Mode***
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Pulse***
  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Polarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::IdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

28.1.6 HRTIM_SimplePWMChannelCfgTypeDef

HRTIM_SimplePWMChannelCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Pulse***

- ***uint32_t Polarity***
- ***uint32_t IdleLevel***

Field Documentation

- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Pulse***
  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Polarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::IdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

28.1.7 HRTIM_SimpleCaptureChannelCfgTypeDef

HRTIM_SimpleCaptureChannelCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Event***
- ***uint32_t EventPolarity***
- ***uint32_t EventSensitivity***
- ***uint32_t EventFilter***

Field Documentation

- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::Event***
  Specifies the external event triggering the capture. This parameter can be any 'EEVx' value of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventPolarity***
  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventSensitivity***
  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventFilter***
  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

28.1.8 HRTIM_SimpleOnePulseChannelCfgTypeDef

HRTIM_SimpleOnePulseChannelCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputIdleLevel***
- ***uint32_t Event***
- ***uint32_t EventPolarity***
- ***uint32_t EventSensitivity***
- ***uint32_t EventFilter***

Field Documentation

- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Pulse***
  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputPolarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***

- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputIdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Event***
  Specifies the external event triggering the pulse generation. This parameter can be any 'EEVx' value of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventPolarity***
  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventSensitivity***
  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***.
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventFilter***
  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

28.1.9 HRTIM_TimerCfgTypeDef

HRTIM_TimerCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t InterruptRequests***
- ***uint32_t DMARequests***
- ***uint32_t DMASrcAddress***
- ***uint32_t DMADstAddress***
- ***uint32_t DMASize***
- ***uint32_t HalfModeEnable***
- ***uint32_t InterleavedMode***
- ***uint32_t StartOnSync***
- ***uint32_t ResetOnSync***
- ***uint32_t DACSynchro***
- ***uint32_t PreloadEnable***
- ***uint32_t UpdateGating***
- ***uint32_t BurstMode***
- ***uint32_t RepetitionUpdate***
- ***uint32_t PushPull***
- ***uint32_t FaultEnable***
- ***uint32_t FaultLock***
- ***uint32_t DeadTimeInsertion***
- ***uint32_t DelayedProtectionMode***
- ***uint32_t BalancedIdleAutomaticResume***
- ***uint32_t UpdateTrigger***
- ***uint32_t ResetTrigger***
- ***uint32_t ResetUpdate***
- ***uint32_t ReSyncUpdate***

Field Documentation

- ***uint32_t HRTIM_TimerCfgTypeDef::InterruptRequests***
  Relevant for all HRTIM timers, including the master. Specifies which interrupts requests must enabled for the timer. This parameter can be any combination of ***HRTIM_Master_Interrupt_Enable*** or ***HRTIM_Timing_Unit_Interrupt_Enable***

- ***uint32_t HRTIM_TimerCfgTypeDef::DMARequests***

  Relevant for all HRTIM timers, including the master. Specifies which DMA requests must be enabled for the timer. This parameter can be any combination of ***HRTIM_Master_DMA_Request_Enable*** or ***HRTIM_Timing_Unit_DMA_Request_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::DMASrcAddress***

  Relevant for all HRTIM timers, including the master. Specifies the address of the source address of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::DMADstAddress***

  Relevant for all HRTIM timers, including the master. Specifies the address of the destination address of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::DMASize***

  Relevant for all HRTIM timers, including the master. Specifies the size of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::HalfModeEnable***

  Relevant for all HRTIM timers, including the master. Specifies whether or not half mode is enabled This parameter can be any value of ***HRTIM_Half_Mode_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::InterleavedMode***

  Relevant for all HRTIM timers, including the master. Specifies whether or not half mode is enabled This parameter can be any value of ***HRTIM_Interleaved_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::StartOnSync***

  Relevant for all HRTIM timers, including the master. Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled). This parameter can be any value of ***HRTIM_Start_On_Sync_Input_Event***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetOnSync***

  Relevant for all HRTIM timers, including the master. Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled). This parameter can be any value of ***HRTIM_Reset_On_Sync_Input_Event***
- ***uint32_t HRTIM_TimerCfgTypeDef::DACSynchro***

  Relevant for all HRTIM timers, including the master. Indicates whether or not the a DAC synchronization event is generated. This parameter can be any value of ***HRTIM_DAC_Synchronization***
- ***uint32_t HRTIM_TimerCfgTypeDef::PreloadEnable***

  Relevant for all HRTIM timers, including the master. Specifies whether or not register preload is enabled. This parameter can be any value of ***HRTIM_Register_Preload_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::UpdateGating***

  Relevant for all HRTIM timers, including the master. Specifies how the update occurs with respect to a burst DMA transaction or update enable inputs (Slave timers only). This parameter can be any value of ***HRTIM_Update_Gating***
- ***uint32_t HRTIM_TimerCfgTypeDef::BurstMode***

  Relevant for all HRTIM timers, including the master. Specifies how the timer behaves during a burst mode operation. This parameter can be any value of ***HRTIM_Timer_Burst_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::RepetitionUpdate***

  Relevant for all HRTIM timers, including the master. Specifies whether or not registers update is triggered by the repetition event. This parameter can be any value of ***HRTIM_Timer_Repetition_Update***
- ***uint32_t HRTIM_TimerCfgTypeDef::PushPull***

  Relevant for Timer A to Timer F. Specifies whether or not the push-pull mode is enabled. This parameter can be any value of ***HRTIM_Timer_Push_Pull_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::FaultEnable***

  Relevant for Timer A to Timer F. Specifies which fault channels are enabled for the timer. This parameter can be a combination of ***HRTIM_Timer_Fault_Enabling***
- ***uint32_t HRTIM_TimerCfgTypeDef::FaultLock***

  Relevant for Timer A to Timer F. Specifies whether or not fault enabling status is write protected. This parameter can be a value of ***HRTIM_Timer_Fault_Lock***

- ***uint32_t HRTIM_TimerCfgTypeDef::DeadTimeInsertion***

  Relevant for Timer A to Timer F. Specifies whether or not dead-time insertion is enabled for the timer. This parameter can be a value of ***HRTIM_Timer_Deadtime_Insertion***
- ***uint32_t HRTIM_TimerCfgTypeDef::DelayedProtectionMode***

  Relevant for Timer A to Timer F. Specifies the delayed protection mode. This parameter can be a value of ***HRTIM_Timer_Delayed_Protection_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::BalancedIdleAutomaticResume***

  Indicates whether or not outputs are automatically re-enabled after a balanced idle event. This parameters can be any value of ***HRTIM_Output_Balanced_Idle_Auto_Resume***
- ***uint32_t HRTIM_TimerCfgTypeDef::UpdateTrigger***

  Relevant for Timer A to Timer F. Specifies source(s) triggering the timer registers update. This parameter can be a combination of ***HRTIM_Timer_Update_Trigger***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetTrigger***

  Relevant for Timer A to Timer F. Specifies source(s) triggering the timer counter reset. This parameter can be a combination of ***HRTIM_Timer_Reset_Trigger***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetUpdate***

  Relevant for Timer A to Timer F. Specifies whether or not registers update is triggered when the timer counter is reset. This parameter can be a value of ***HRTIM_Timer_Reset_Update***
- ***uint32_t HRTIM_TimerCfgTypeDef::ReSyncUpdate***

  Relevant for Timer A to Timer F. Specifies whether update source is coming from the timing unit ***HRTIM_Timer_ReSyncUpdate***

28.1.10 HRTIM_TimerCtlTypeDef

HRTIM_TimerCtlTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t UpDownMode***
- ***uint32_t TrigHalf***
- ***uint32_t GreaterCMP3***
- ***uint32_t GreaterCMP1***
- ***uint32_t DualChannelDacReset***
- ***uint32_t DualChannelDacStep***
- ***uint32_t DualChannelDacEnable***

Field Documentation

- ***uint32_t HRTIM_TimerCtlTypeDef::UpDownMode***

  Relevant for Timer A to Timer F. Specifies whether or not counter is operating in up or up-down counting mode. This parameter can be a value of ***HRTIM_Timer_UpDown_Mode***
- ***uint32_t HRTIM_TimerCtlTypeDef::TrigHalf***

  Relevant for Timer A to Timer F. Specifies whether or not compare 2 is operating in Trigger half mode. This parameter can be a value of ***HRTIM_Timer_TrigHalf_Mode***
- ***uint32_t HRTIM_TimerCtlTypeDef::GreaterCMP3***

  Relevant for Timer A to Timer F. Specifies whether or not compare 3 is operating in compare match or greater mode. This parameter can be a value of ***HRTIM_Timer_GreaterCMP3_Mode***
- ***uint32_t HRTIM_TimerCtlTypeDef::GreaterCMP1***

  Relevant for Timer A to Timer F. Specifies whether or not compare 1 is operating in compare match or greater mode. This parameter can be a value of ***HRTIM_Timer_GreaterCMP1_Mode***
- ***uint32_t HRTIM_TimerCtlTypeDef::DualChannelDacReset***

  Relevant for Timer A to Timer F. Specifies how the hrtim_dac_reset_trgx trigger is generated. This parameter can be a value of ***HRTIM_Timer_DualChannelDac_Reset***
- ***uint32_t HRTIM_TimerCtlTypeDef::DualChannelDacStep***

  Relevant for Timer A to Timer F. Specifies how the hrtim_dac_step_trgx trigger is generated. This parameter can be a value of ***HRTIM_Timer_DualChannelDac_Step***

- ***uint32_t HRTIM_TimerCtlTypeDef::DualChannelDacEnable***

  Relevant for Timer A to Timer F. Enables or not the dual channel DAC triggering mechanism. This parameter can be a value of ***HRTIM_Timer_DualChannelDac_Enable***

28.1.11 HRTIM_CompareCfgTypeDef

HRTIM_CompareCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t CompareValue***
- ***uint32_t AutoDelayedMode***
- ***uint32_t AutoDelayedTimeout***

Field Documentation

- ***uint32_t HRTIM_CompareCfgTypeDef::CompareValue***

  Specifies the compare value of the timer compare unit. The minimum value must be greater than or equal to 3 periods of the fHRTIM clock. The maximum value must be less than or equal to 0xFFFFU - 1 periods of the fHRTIM clock
- ***uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedMode***

  Specifies the auto delayed mode for compare unit 2 or 4. This parameter can be a value of ***HRTIM_Compare_Unit_Auto_Delayed_Mode***
- ***uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedTimeout***

  Specifies compare value for timing unit 1 or 3 when auto delayed mode with time out is selected. CompareValue + AutoDelayedTimeout must be less than 0xFFFFU

28.1.12 HRTIM_CaptureValueTypeDef

HRTIM_CaptureValueTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Value***
- ***uint32_t Dir***

Field Documentation

- ***uint32_t HRTIM_CaptureValueTypeDef::Value***

  Holds the counter value when the capture event occurred. This parameter can be a number between 0x0 and 0xFFFFU
- ***uint32_t HRTIM_CaptureValueTypeDef::Dir***

  Holds the counting direction value when the capture event occurred. This parameter can be a value of ***HRTIM_Timer_UpDown_Mode***

28.1.13 HRTIM_CaptureCfgTypeDef

HRTIM_CaptureCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint64_t Trigger***

Field Documentation

- ***uint64_t HRTIM_CaptureCfgTypeDef::Trigger***

  Specifies source(s) triggering the capture. This parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***

28.1.14 HRTIM_OutputCfgTypeDef

HRTIM_OutputCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SetSource***
- ***uint32_t ResetSource***
- ***uint32_t IdleMode***
- ***uint32_t IdleLevel***

- ***uint32_t FaultLevel***
- ***uint32_t ChopperModeEnable***
- ***uint32_t BurstModeEntryDelayed***

Field Documentation

- ***uint32_t HRTIM_OutputCfgTypeDef::Polarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_OutputCfgTypeDef::SetSource***
  Specifies the event(s) transitioning the output from its inactive level to its active level. This parameter can be a combination of ***HRTIM_Output_Set_Source***
- ***uint32_t HRTIM_OutputCfgTypeDef::ResetSource***
  Specifies the event(s) transitioning the output from its active level to its inactive level. This parameter can be a combination of ***HRTIM_Output_Reset_Source***
- ***uint32_t HRTIM_OutputCfgTypeDef::IdleMode***
  Specifies whether or not the output is affected by a burst mode operation. This parameter can be any value of ***HRTIM_Output_Idle_Mode***
- ***uint32_t HRTIM_OutputCfgTypeDef::IdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***
- ***uint32_t HRTIM_OutputCfgTypeDef::FaultLevel***
  Specifies whether the output level is active or inactive when in FAULT state. This parameter can be any value of ***HRTIM_Output_FAULT_Level***
- ***uint32_t HRTIM_OutputCfgTypeDef::ChopperModeEnable***
  Indicates whether or not the chopper mode is enabled This parameter can be any value of ***HRTIM_Output_Chopper_Mode_Enable***
- ***uint32_t HRTIM_OutputCfgTypeDef::BurstModeEntryDelayed***
  Indicates whether or not dead-time is inserted when entering the IDLE state during a burst mode operation. This parameters can be any value of ***HRTIM_Output_Burst_Mode_Entry_Delayed***

28.1.15 HRTIM_TimerEventFilteringCfgTypeDef

HRTIM_TimerEventFilteringCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Filter***
- ***uint32_t Latch***

Field Documentation

- ***uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Filter***
  Specifies the type of event filtering within the timing unit. This parameter can be a value of ***HRTIM_Timer_External_Event_Filter***
- ***uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Latch***
  Specifies whether or not the signal is latched. This parameter can be a value of ***HRTIM_Timer_External_Event_Latch***

28.1.16 HRTIM_DeadTimeCfgTypeDef

HRTIM_DeadTimeCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t RisingValue***
- ***uint32_t RisingSign***
- ***uint32_t RisingLock***
- ***uint32_t RisingSignLock***
- ***uint32_t FallingValue***
- ***uint32_t FallingSign***

- ***uint32_t FallingLock***
- ***uint32_t FallingSignLock***

Field Documentation

- ***uint32_t HRTIM_DeadTimeCfgTypeDef::Prescaler***
  Specifies the dead-time prescaler. This parameter can be a value of ***HRTIM_Deadtime_Prescaler_Ratio***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingValue***
  Specifies the dead-time following a rising edge. This parameter can be a number between 0x0 and 0x1FFU
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSign***
  Specifies whether the dead-time is positive or negative on rising edge. This parameter can be a value of ***HRTIM_Deadtime_Rising_Sign***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingLock***
  Specifies whether or not dead-time rising settings (value and sign) are write protected. This parameter can be a value of ***HRTIM_Deadtime_Rising_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSignLock***
  Specifies whether or not dead-time rising sign is write protected. This parameter can be a value of ***HRTIM_Deadtime_Rising_Sign_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingValue***
  Specifies the dead-time following a falling edge. This parameter can be a number between 0x0 and 0x1FFU
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSign***
  Specifies whether the dead-time is positive or negative on falling edge. This parameter can be a value of ***HRTIM_Deadtime_Falling_Sign***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingLock***
  Specifies whether or not dead-time falling settings (value and sign) are write protected. This parameter can be a value of ***HRTIM_Deadtime_Falling_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSignLock***
  Specifies whether or not dead-time falling sign is write protected. This parameter can be a value of ***HRTIM_Deadtime_Falling_Sign_Lock***

28.1.17 HRTIM_ChopperModeCfgTypeDef

HRTIM_ChopperModeCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t CarrierFreq***
- ***uint32_t DutyCycle***
- ***uint32_t StartPulse***

Field Documentation

- ***uint32_t HRTIM_ChopperModeCfgTypeDef::CarrierFreq***
  Specifies the Timer carrier frequency value. This parameter can be a value of ***HRTIM_Chopper_Frequency***
- ***uint32_t HRTIM_ChopperModeCfgTypeDef::DutyCycle***
  Specifies the Timer chopper duty cycle value. This parameter can be a value of ***HRTIM_Chopper_Duty_Cycle***
- ***uint32_t HRTIM_ChopperModeCfgTypeDef::StartPulse***
  Specifies the Timer pulse width value. This parameter can be a value of ***HRTIM_Chopper_Start_Pulse_Width***

28.1.18 HRTIM_EventCfgTypeDef

HRTIM_EventCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t Sensitivity***
- ***uint32_t Filter***

- ***uint32_t FastMode***

Field Documentation

- ***uint32_t HRTIM_EventCfgTypeDef::Source***
  Identifies the source of the external event. This parameter can be a value of ***HRTIM_External_Event_Sources***
- ***uint32_t HRTIM_EventCfgTypeDef::Polarity***
  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_EventCfgTypeDef::Sensitivity***
  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***
- ***uint32_t HRTIM_EventCfgTypeDef::Filter***
  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***
- ***uint32_t HRTIM_EventCfgTypeDef::FastMode***
  Indicates whether or not low latency mode is enabled for the external event. This parameter can be a value of ***HRTIM_External_Event_Fast_Mode***

28.1.19 HRTIM_FaultCfgTypeDef

HRTIM_FaultCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t Filter***
- ***uint32_t Lock***

Field Documentation

- ***uint32_t HRTIM_FaultCfgTypeDef::Source***
  Identifies the source of the fault. This parameter can be a value of ***HRTIM_Fault_Sources***
- ***uint32_t HRTIM_FaultCfgTypeDef::Polarity***
  Specifies the polarity of the fault event. This parameter can be a value of ***HRTIM_Fault_Polarity***
- ***uint32_t HRTIM_FaultCfgTypeDef::Filter***
  Defines the frequency used to sample the Fault input and the length of the digital filter. This parameter can be a value of ***HRTIM_Fault_Filter***
- ***uint32_t HRTIM_FaultCfgTypeDef::Lock***
  Indicates whether or not fault programming bits are write protected. This parameter can be a value of ***HRTIM_Fault_Lock***

28.1.20 HRTIM_FaultBlankingCfgTypeDef

HRTIM_FaultBlankingCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Threshold***
- ***uint32_t ResetMode***
- ***uint32_t BlankingSource***

Field Documentation

- ***uint32_t HRTIM_FaultBlankingCfgTypeDef::Threshold***
  Specifies the Fault counter Threshold. This parameter can be a number between 0x0 and 0xF
- ***uint32_t HRTIM_FaultBlankingCfgTypeDef::ResetMode***
  Specifies the reset mode of a fault event counter. This parameter can be a value of ***HRTIM_Fault_ResetMode***
- ***uint32_t HRTIM_FaultBlankingCfgTypeDef::BlankingSource***
  Specifies the blanking source of a fault event. This parameter can be a value of ***HRTIM_Fault_Blanking***

28.1.21 HRTIM_BurstModeCfgTypeDef

HRTIM_BurstModeCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t PreloadEnable***
- ***uint32_t Trigger***
- ***uint32_t IdleDuration***
- ***uint32_t Period***

Field Documentation

- ***uint32_t HRTIM_BurstModeCfgTypeDef::Mode***

  Specifies the burst mode operating mode. This parameter can be a value of ***HRTIM_Burst_Mode_Operating_Mode***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::ClockSource***

  Specifies the burst mode clock source. This parameter can be a value of ***HRTIM_Burst_Mode_Clock_Source***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Prescaler***

  Specifies the burst mode prescaler. This parameter can be a value of ***HRTIM_Burst_Mode_Prescaler***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::PreloadEnable***

  Specifies whether or not preload is enabled for burst mode related registers (HRTIM_BMCMPR and HRTIM_BMPER). This parameter can be a combination of ***HRTIM_Burst_Mode_Register_Preload_Enable***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Trigger***

  Specifies the event(s) triggering the burst operation. This parameter can be a combination of ***HRTIM_Burst_Mode_Trigger***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::IdleDuration***

  Specifies number of periods during which the selected timers are in idle state. This parameter can be a number between 0x0 and 0xFFFF
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Period***

  Specifies burst mode repetition period. This parameter can be a number between 0x1 and 0xFFFF

28.1.22 HRTIM_ADCTriggerCfgTypeDef

HRTIM_ADCTriggerCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t UpdateSource***
- ***uint32_t Trigger***

Field Documentation

- ***uint32_t HRTIM_ADCTriggerCfgTypeDef::UpdateSource***

  Specifies the ADC trigger update source. This parameter can be a value of ***HRTIM_ADC_Trigger_Update_Source***
- ***uint32_t HRTIM_ADCTriggerCfgTypeDef::Trigger***

  Specifies the event(s) triggering the ADC conversion. This parameter can be a combination of ***HRTIM_ADC_Trigger_Event***

28.1.23 HRTIM_ExternalEventCfgTypeDef

HRTIM_ExternalEventCfgTypeDef is defined in the stm32g4xx_hal_hrtim.h

Data Fields

- ***uint32_t ResetMode***
- ***uint32_t Source***
- ***uint32_t Counter***

Field Documentation

- ***uint32_t HRTIM_ExternalEventCfgTypeDef::ResetMode***

  Specifies the External Event Counter A or B Reset Mode. This parameter can be a value of ***HRTIM_Timer_External_Event_ResetMode***
- ***uint32_t HRTIM_ExternalEventCfgTypeDef::Source***

  Specifies the External Event Counter source selection. This parameter can be one of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_ExternalEventCfgTypeDef::Counter***

  Specifies the External Event Counter Threshold. This parameter can be a number between 0x0 and 0x3F

28.2 HRTIM Firmware driver API description

The following section lists the various functions of the HRTIM library.

28.2.1 Simple mode v.s. waveform mode

The HRTIM HAL API is split into 2 categories:

1. Simple functions: these functions allow for using a HRTIM timer as a general purpose timer with high resolution capabilities. HRTIM simple modes are managed through the set of functions named HAL_HRTIM_Simple<Function>. These functions are similar in name and usage to the one defined for the TIM peripheral. When a HRTIM timer operates in simple mode, only a very limited set of HRTIM features are used. Following simple modes are proposed:
   - Output compare mode,
   - PWM output mode,
   - Input capture mode,
   - One pulse mode.
2. Waveform functions: These functions allow taking advantage of the HRTIM flexibility to produce numerous types of control signal. When a HRTIM timer operates in waveform mode, all the HRTIM features are accessible without any restriction. HRTIM waveform modes are managed through the set of functions named HAL_HRTIM_Waveform<Function>

28.2.2 How to use this driver

1. Initialize the HRTIM low level resources by implementing the HAL_HRTIM_MspInit() function:
   a. Enable the HRTIM clock source using __HRTIMx_CLK_ENABLE()
   b. Connect HRTIM pins to MCU I/Os
      - Enable the clock for the HRTIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE()
      - Configure these GPIO pins in Alternate Function mode using HAL_GPIO_Init()
   c. When using DMA to control data transfer (e.g HAL_HRTIM_SimpleBaseStart_DMA())
      - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      - Initialize the DMA handle
      - Associate the initialized DMA handle to the appropriate DMA handle of the HRTIM handle using __HAL_LINKDMA()
      - Initialize the DMA channel using HAL_DMA_Init()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA channel using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
   d. In case of using interrupt mode (e.g HAL_HRTIM_SimpleBaseStart_IT())
      - Configure the priority and enable the NVIC for the concerned HRTIM interrupt using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HRTIM HAL using HAL_HRTIM_Init(). The HRTIM configuration structure (field of the HRTIM handle) specifies which global interrupt of whole HRTIM must be enabled (Burst mode period, System fault, Faults). It also contains the HRTIM external synchronization configuration. HRTIM can act as a master (generating a synchronization signal) or as a slave (waiting for a trigger to be synchronized).

3. Start the high resolution unit using HAL_HRTIM_DLLCalibrationStart(). DLL calibration is executed periodically and compensate for potential voltage and temperature drifts. DLL calibration period is specified by the CalibrationRate argument.
4. HRTIM timers cannot be used until the high resolution unit is ready. This can be checked using HAL_HRTIM_PollForDLLCalibration(): this function returns HAL_OK if DLL calibration is completed or HAL_TIMEOUT if the DLL calibration is still going on when timeout given as argument expires. DLL calibration can also be started in interrupt mode using HAL_HRTIM_DLLCalibrationStart_IT(). In that case an interrupt is generated when the DLL calibration is completed. Note that as DLL calibration is executed on a periodic basis an interrupt will be generated at the end of every DLL calibration operation (worst case: one interrupt every 14 micro seconds !).
5. Configure HRTIM resources shared by all HRTIM timers
   a. Burst Mode Controller:
      - HAL_HRTIM_BurstModeConfig(): configures the HRTIM burst mode controller: operating mode (continuous or one-shot mode), clock (source, prescaler) , trigger(s), period, idle duration.
   b. External Events Conditioning:
      - HAL_HRTIM_EventConfig(): configures the conditioning of an external event channel: source, polarity, edge-sensitivity. External event can be used as triggers (timer reset, input capture, burst mode, ADC triggers, delayed protection) They can also be used to set or reset timer outputs. Up to 10 event channels are available.
      - HAL_HRTIM_EventPrescalerConfig(): configures the external event sampling clock (used for digital filtering).
   c. Fault Conditioning:
      - HAL_HRTIM_FaultConfig(): configures the conditioning of a fault channel: source, polarity, edge-sensitivity. Fault channels are used to disable the outputs in case of an abnormal operation. Up to 6 fault channels are available.
      - HAL_HRTIM_FaultPrescalerConfig(): configures the fault sampling clock (used for digital filtering).
      - HAL_HRTIM_FaultModeCtl(): Enables or disables fault input(s) circuitry. By default all fault inputs are disabled.
   d. ADC trigger:
      - HAL_HRTIM_ADCTriggerConfig(): configures the source triggering the update of the ADC trigger register and the ADC trigger. 4 independent triggers are available to start both the regular and the injected sequencers of the 2 ADCs
6. Configure HRTIM timer time base using HAL_HRTIM_TimeBaseConfig(). This function must be called whatever the HRTIM timer operating mode is (simple v.s. waveform). It configures mainly:
   a. The HRTIM timer counter operating mode (continuous v.s. one shot)
   b. The HRTIM timer clock prescaler
   c. The HRTIM timer period
   d. The HRTIM timer repetition counter

If the HRTIM timer operates in simple mode

1. Start or Stop simple timers
   - Simple time base: HAL_HRTIM_SimpleBaseStart(),HAL_HRTIM_SimpleBaseStop(), HAL_HRTIM_SimpleBaseStart_IT(),HAL_HRTIM_SimpleBaseStop_IT(), HAL_HRTIM_SimpleBaseStart_DMA(),HAL_HRTIM_SimpleBaseStop_DMA().
   - Simple output compare: HAL_HRTIM_SimpleOCChannelConfig(), HAL_HRTIM_SimpleOCStart(),HAL_HRTIM_SimpleOCStop(), HAL_HRTIM_SimpleOCStart_IT(),HAL_HRTIM_SimpleOCStop_IT(), HAL_HRTIM_SimpleOCStart_DMA(),HAL_HRTIM_SimpleOCStop_DMA(),
   - Simple PWM output: HAL_HRTIM_SimplePWMChannelConfig(), HAL_HRTIM_SimplePWMStart(),HAL_HRTIM_SimplePWMStop(), HAL_HRTIM_SimplePWMStart_IT(),HAL_HRTIM_SimplePWMStop_IT(), HAL_HRTIM_SimplePWMStart_DMA(),HAL_HRTIM_SimplePWMStop_DMA(),
   - Simple input capture: HAL_HRTIM_SimpleCaptureChannelConfig(), HAL_HRTIM_SimpleCaptureStart(),HAL_HRTIM_SimpleCaptureStop(), HAL_HRTIM_SimpleCaptureStart_IT(),HAL_HRTIM_SimpleCaptureStop_IT(), HAL_HRTIM_SimpleCaptureStart_DMA(),HAL_HRTIM_SimpleCaptureStop_DMA().
   - Simple one pulse: HAL_HRTIM_SimpleOnePulseChannelConfig(), HAL_HRTIM_SimpleOnePulseStart(),HAL_HRTIM_SimpleOnePulseStop(), HAL_HRTIM_SimpleOnePulseStart_IT(),HAL_HRTIM_SimpleOnePulseStop_It().

If the HRTIM timer operates in waveform mode

1. Completes waveform timer configuration
   - HAL_HRTIM_WaveformTimerConfig(): configuration of a HRTIM timer operating in wave form mode mainly consists in:
     - Enabling the HRTIM timer interrupts and DMA requests.
     - Enabling the half mode for the HRTIM timer.
     - Defining how the HRTIM timer reacts to external synchronization input.
     - Enabling the push-pull mode for the HRTIM timer.
     - Enabling the fault channels for the HRTIM timer.
     - Enabling the dead-time insertion for the HRTIM timer.
     - Setting the delayed protection mode for the HRTIM timer (source and outputs on which the delayed protection are applied).
     - Specifying the HRTIM timer update and reset triggers.
     - Specifying the HRTIM timer registers update policy (e.g. pre-load enabling).
   - HAL_HRTIM_TimerEventFilteringConfig(): configures external event blanking and windowing circuitry of a HRTIM timer:
     - Blanking: to mask external events during a defined time period a defined time period
     - Windowing, to enable external events only during a defined time period
   - HAL_HRTIM_DeadTimeConfig(): configures the dead-time insertion unit for a HRTIM timer. Allows to generate a couple of complementary signals from a single reference waveform, with programmable delays between active state.
   - HAL_HRTIM_ChopperModeConfig(): configures the parameters of the high-frequency carrier signal added on top of the timing unit output. Chopper mode can be enabled or disabled for each timer output separately (see HAL_HRTIM_WaveformOutputConfig()).
   - HAL_HRTIM_BurstDMAConfig(): configures the burst DMA burst controller. Allows having multiple HRTIM registers updated with a single DMA request. The burst DMA operation is started by calling HAL_HRTIM_BurstDMATransfer().
   - HAL_HRTIM_WaveformCompareConfig():configures the compare unit of a HRTIM timer. This operation consists in setting the compare value and possibly specifying the auto delayed mode for compare units 2 and 4 (allows to have compare events generated relatively to capture events). Note that when auto delayed mode is needed, the capture unit associated to the compare unit must be configured separately.
   - HAL_HRTIM_WaveformCaptureConfig(): configures the capture unit of a HRTIM timer. This operation consists in specifying the source(s) triggering the capture (timer register update event, external event, timer output set/reset event, other HRTIM timer related events).
   - HAL_HRTIM_WaveformOutputConfig(): configuration of a HRTIM timer output mainly consists in:
     - Setting the output polarity (active high or active low),
     - Defining the set/reset crossbar for the output,
     - Specifying the fault level (active or inactive) in IDLE and FAULT states.,
2. Set waveform timer output(s) level
   - HAL_HRTIM_WaveformSetOutputLevel(): forces the output to its active or inactive level. For example, when deadtime insertion is enabled it is necessary to force the output level by software to have the outputs in a complementary state as soon as the RUN mode is entered.
3. Enable or Disable waveform timer output(s)
   - HAL_HRTIM_WaveformOutputStart(),HAL_HRTIM_WaveformOutputStop().
4. Start or Stop waveform HRTIM timer(s).
   - HAL_HRTIM_WaveformCountStart(),HAL_HRTIM_WaveformCountStop(),
   - HAL_HRTIM_WaveformCountStart_IT(),HAL_HRTIM_WaveformCountStop_IT(),
   - HAL_HRTIM_WaveformCountStart_DMA(),HAL_HRTIM_WaveformCountStop_DMA(),
5. Burst mode controller enabling:
   - HAL_HRTIM_BurstModeCtl(): activates or de-activates the burst mode controller.

6. Some HRTIM operations can be triggered by software:
   - HAL_HRTIM_BurstModeSoftwareTrigger(): calling this function trigs the burst operation.
   - HAL_HRTIM_SoftwareCapture(): calling this function trigs the capture of the HRTIM timer counter.
   - HAL_HRTIM_SoftwareUpdate(): calling this function trigs the update of the pre-loadable registers of the HRTIM timer
   - HAL_HRTIM_SoftwareReset():calling this function resets the HRTIM timer counter.
7. Some functions can be used any time to retrieve HRTIM timer related information
   - HAL_HRTIM_GetCapturedValue(): returns actual value of the capture register of the designated capture unit.
   - HAL_HRTIM_WaveformGetOutputLevel(): returns actual level (ACTIVE/INACTIVE) of the designated timer output.
   - HAL_HRTIM_WaveformGetOutputState():returns actual state (IDLE/RUN/FAULT) of the designated timer output.
   - HAL_HRTIM_GetDelayedProtectionStatus():returns actual level (ACTIVE/INACTIVE) of the designated output when the delayed protection was triggered.
   - HAL_HRTIM_GetBurstStatus(): returns the actual status (ACTIVE/INACTIVE) of the burst mode controller.
   - HAL_HRTIM_GetCurrentPushPullStatus(): when the push-pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the push-pull status indicates on which output the signal is currently active (e.g signal applied on output 1 and output 2 forced inactive or vice versa).
   - HAL_HRTIM_GetIdlePushPullStatus(): when the push-pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the idle push-pull status indicates during which period the delayed protection request occurred (e.g. protection occurred when the output 1 was active and output 2 forced inactive or vice versa).
8. Some functions can be used any time to retrieve actual HRTIM status
   - HAL_HRTIM_GetState(): returns actual HRTIM instance HAL state.

Callback registration

The compilation flag USE_HAL_HRTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_HRTIM_RegisterCallback() or HAL_HRTIM_TIMxRegisterCallback() to register an interrupt callback.

Function HAL_HRTIM_RegisterCallback() allows to register following callbacks:

- Fault1Callback : Fault 1 interrupt callback function
- Fault2Callback : Fault 2 interrupt callback function
- Fault3Callback : Fault 3 interrupt callback function
- Fault4Callback : Fault 4 interrupt callback function
- Fault5Callback : Fault 5 interrupt callback function
- Fault6Callback : Fault 6 interrupt callback function
- SystemFaultCallback : System fault interrupt callback function
- DLLCalibrationReadyCallback : DLL Ready interrupt callback function
- BurstModePeriodCallback : Burst mode period interrupt callback function
- SynchronizationEventCallback : Sync Input interrupt callback function
- ErrorCallback : DMA error callback function
- MspInitCallback : HRTIM MspInit callback function
- MspDeInitCallback : HRTIM MspInit callback function

Function HAL_HRTIM_TIMxRegisterCallback() allows to register following callbacks:

- RegistersUpdateCallback : Timer x Update interrupt callback function
- RepetitionEventCallback : Timer x Repetition interrupt callback function
- Compare1EventCallback : Timer x Compare 1 match interrupt callback function
- Compare2EventCallback : Timer x Compare 2 match interrupt callback function
- Compare3EventCallback : Timer x Compare 3 match interrupt callback function

- Compare4EventCallback : Timer x Compare 4 match interrupt callback function
- Capture1EventCallback : Timer x Capture 1 interrupts callback function
- Capture2EventCallback : Timer x Capture 2 interrupts callback function
- DelayedProtectionCallback : Timer x Delayed protection interrupt callback function
- CounterResetCallback : Timer x counter reset/roll-over interrupt callback function
- Output1SetCallback : Timer x output 1 set interrupt callback function
- Output1ResetCallback : Timer x output 1 reset interrupt callback function
- Output2SetCallback : Timer x output 2 set interrupt callback function
- Output2ResetCallback : Timer x output 2 reset interrupt callback function
- BurstDMATransferCallback : Timer x Burst DMA completed interrupt callback function

Both functions take as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_HRTIM_UnRegisterCallback or HAL_HRTIM_TIMxUnRegisterCallback to reset a callback to the default weak function. Both functions take as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_HRTIM_Init() and when the state is HAL_HRTIM_STATE_RESET all callbacks are set to the corresponding weak functions (e.g HAL_HRTIM_Fault1Callback) Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_HRTIM_Init()/ HAL_HRTIM_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_HRTIM_Init()/ HAL_HRTIM_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_HRTIM_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_HRTIM_STATE_READY or HAL_HRTIM_STATE_RESET states, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_HRTIM_RegisterCallback() before calling HAL_HRTIM_DeInit() or HAL_HRTIM_Init() function.

When the compilation flag USE_HAL_HRTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

28.2.3 Initialization and Time Base Configuration functions

This section provides functions allowing to:

- Initialize a HRTIM instance
- De-initialize a HRTIM instance
- Initialize the HRTIM MSP
- De-initialize the HRTIM MSP
- Start the high-resolution unit (start DLL calibration)
- Check that the high resolution unit is ready (DLL calibration done)
- Configure the time base unit of a HRTIM timer

This section contains the following APIs:

- ***HAL_HRTIM_Init***
- ***HAL_HRTIM_DeInit***
- ***HAL_HRTIM_MspInit***
- ***HAL_HRTIM_MspDeInit***
- ***HAL_HRTIM_DLLCalibrationStart***
- ***HAL_HRTIM_DLLCalibrationStart_IT***
- ***HAL_HRTIM_PollForDLLCalibration***
- ***HAL_HRTIM_TimeBaseConfig***

28.2.4 Simple time base mode functions

This section provides functions allowing to:

- Start simple time base
- Stop simple time base
- Start simple time base and enable interrupt

- Stop simple time base and disable interrupt
- Start simple time base and enable DMA transfer
- Stop simple time base and disable DMA transfer

Note: When a HRTIM timer operates in simple time base mode, the timer counter counts from 0 to the period value.

This section contains the following APIs:

- ***HAL_HRTIM_SimpleBaseStart***
- ***HAL_HRTIM_SimpleBaseStop***
- ***HAL_HRTIM_SimpleBaseStart_IT***
- ***HAL_HRTIM_SimpleBaseStop_IT***
- ***HAL_HRTIM_SimpleBaseStart_DMA***
- ***HAL_HRTIM_SimpleBaseStop_DMA***

28.2.5 Simple output compare functions

This section provides functions allowing to:

- Configure simple output channel
- Start simple output compare
- Stop simple output compare
- Start simple output compare and enable interrupt
- Stop simple output compare and disable interrupt
- Start simple output compare and enable DMA transfer
- Stop simple output compare and disable DMA transfer

Note: When a HRTIM timer operates in simple output compare mode the output level is set to a programmable value when a match is found between the compare register and the counter. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2

This section contains the following APIs:

- ***HAL_HRTIM_SimpleOCChannelConfig***
- ***HAL_HRTIM_SimpleOCStart***
- ***HAL_HRTIM_SimpleOCStop***
- ***HAL_HRTIM_SimpleOCStart_IT***
- ***HAL_HRTIM_SimpleOCStop_IT***
- ***HAL_HRTIM_SimpleOCStart_DMA***
- ***HAL_HRTIM_SimpleOCStop_DMA***

28.2.6 Simple PWM output functions

This section provides functions allowing to:

- Configure simple PWM output channel
- Start simple PWM output
- Stop simple PWM output
- Start simple PWM output and enable interrupt
- Stop simple PWM output and disable interrupt
- Start simple PWM output and enable DMA transfer
- Stop simple PWM output and disable DMA transfer

Note: When a HRTIM timer operates in simple PWM output mode the output level is set to a programmable value when a match is found between the compare register and the counter and reset when the timer period is reached. Duty cycle is determined by the comparison value. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2

This section contains the following APIs:

- ***HAL_HRTIM_SimplePWMChannelConfig***
- ***HAL_HRTIM_SimplePWMStart***

- *HAL_HRTIM_SimplePWMStop*
- *HAL_HRTIM_SimplePWMStart_IT*
- *HAL_HRTIM_SimplePWMStop_IT*
- *HAL_HRTIM_SimplePWMStart_DMA*
- *HAL_HRTIM_SimplePWMStop_DMA*

28.2.7 Simple input capture functions

This section provides functions allowing to:

- Configure simple input capture channel
- Start simple input capture
- Stop simple input capture
- Start simple input capture and enable interrupt
- Stop simple input capture and disable interrupt
- Start simple input capture and enable DMA transfer
- Stop simple input capture and disable DMA transfer

Note: When a HRTIM timer operates in simple input capture mode the Capture Register (HRTIM_CPT1/2xR) is used to latch the value of the timer counter counter after a transition detected on a given external event input.

This section contains the following APIs:

- *HAL_HRTIM_SimpleCaptureChannelConfig*
- *HAL_HRTIM_SimpleCaptureStart*
- *HAL_HRTIM_SimpleCaptureStop*
- *HAL_HRTIM_SimpleCaptureStart_IT*
- *HAL_HRTIM_SimpleCaptureStop_IT*
- *HAL_HRTIM_SimpleCaptureStart_DMA*
- *HAL_HRTIM_SimpleCaptureStop_DMA*

28.2.8 Simple one pulse functions

This section provides functions allowing to:

- Configure one pulse channel
- Start one pulse generation
- Stop one pulse generation
- Start one pulse generation and enable interrupt
- Stop one pulse generation and disable interrupt

Note: When a HRTIM timer operates in simple one pulse mode the timer counter is started in response to transition detected on a given external event input to generate a pulse with a programmable length after a programmable delay.

This section contains the following APIs:

- *HAL_HRTIM_SimpleOnePulseChannelConfig*
- *HAL_HRTIM_SimpleOnePulseStart*
- *HAL_HRTIM_SimpleOnePulseStop*
- *HAL_HRTIM_SimpleOnePulseStart_IT*
- *HAL_HRTIM_SimpleOnePulseStop_IT*

28.2.9 HRTIM configuration functions

This section provides functions allowing to configure the HRTIM resources shared by all the HRTIM timers operating in waveform mode:

- Configure the burst mode controller
- Configure an external event conditioning
- Configure the external events sampling clock
- Configure a fault conditioning

- Enable or disable fault inputs
- Configure the faults sampling clock
- Configure an ADC trigger

This section contains the following APIs:

- *HAL_HRTIM_BurstModeConfig*
- *HAL_HRTIM_EventConfig*
- *HAL_HRTIM_EventPrescalerConfig*
- *HAL_HRTIM_FaultConfig*
- *HAL_HRTIM_FaultPrescalerConfig*
- *HAL_HRTIM_FaultBlankingConfigAndEnable*
- *HAL_HRTIM_FaultCounterConfig*
- *HAL_HRTIM_FaultCounterReset*
- *HAL_HRTIM_FaultModeCtl*
- *HAL_HRTIM_ADCTriggerConfig*
- *HAL_HRTIM_ADCPostScalerConfig*
- *HAL_HRTIM_RollOverModeConfig*
- *HAL_HRTIM_SwapTimerOutput*
- *HAL_HRTIM_OutputSwapEnable*
- *HAL_HRTIM_OutputSwapDisable*

28.2.10 HRTIM timer configuration and control functions

This section provides functions used to configure and control a HRTIM timer operating in waveform mode:

- Configure HRTIM timer general behavior
- Configure HRTIM timer event filtering
- Configure HRTIM timer deadtime insertion
- Configure HRTIM timer chopper mode
- Configure HRTIM timer burst DMA
- Configure HRTIM timer compare unit
- Configure HRTIM timer capture unit
- Configure HRTIM timer output
- Set HRTIM timer output level
- Enable HRTIM timer output
- Disable HRTIM timer output
- Start HRTIM timer
- Stop HRTIM timer
- Start HRTIM timer and enable interrupt
- Stop HRTIM timer and disable interrupt
- Start HRTIM timer and enable DMA transfer
- Stop HRTIM timer and disable DMA transfer
- Enable or disable the burst mode controller
- Start the burst mode controller (by software)
- Trigger a Capture (by software)
- Update the HRTIM timer preloadable registers (by software)
- Reset the HRTIM timer counter (by software)
- Start a burst DMA transfer
- Enable timer register update
- Disable timer register update

This section contains the following APIs:

- *HAL_HRTIM_WaveformTimerConfig*

- *HAL_HRTIM_WaveformTimerControl*
- *HAL_HRTIM_TimerDualChannelDacConfig*
- *HAL_HRTIM_TimerEventFilteringConfig*
- *HAL_HRTIM_ExtEventCounterConfig*
- *HAL_HRTIM_ExtEventCounterEnable*
- *HAL_HRTIM_ExtEventCounterDisable*
- *HAL_HRTIM_ExtEventCounterReset*
- *HAL_HRTIM_DeadTimeConfig*
- *HAL_HRTIM_ChopperModeConfig*
- *HAL_HRTIM_BurstDMAConfig*
- *HAL_HRTIM_WaveformCompareConfig*
- *HAL_HRTIM_WaveformCaptureConfig*
- *HAL_HRTIM_WaveformOutputConfig*
- *HAL_HRTIM_WaveformSetOutputLevel*
- *HAL_HRTIM_WaveformOutputStart*
- *HAL_HRTIM_WaveformOutputStop*
- *HAL_HRTIM_WaveformCountStart*
- *HAL_HRTIM_WaveformCountStop*
- *HAL_HRTIM_WaveformCountStart_IT*
- *HAL_HRTIM_WaveformCountStop_IT*
- *HAL_HRTIM_WaveformCountStart_DMA*
- *HAL_HRTIM_WaveformCountStop_DMA*
- *HAL_HRTIM_BurstModeCtl*
- *HAL_HRTIM_BurstModeSoftwareTrigger*
- *HAL_HRTIM_SoftwareCapture*
- *HAL_HRTIM_SoftwareUpdate*
- *HAL_HRTIM_SwapTimerOutput*
- *HAL_HRTIM_SoftwareReset*
- *HAL_HRTIM_OutputSwapEnable*
- *HAL_HRTIM_OutputSwapDisable*
- *HAL_HRTIM_BurstDMATransfer*
- *HAL_HRTIM_UpdateEnable*
- *HAL_HRTIM_UpdateDisable*

28.2.11 Peripheral State functions

This section provides functions used to get HRTIM or HRTIM timer specific information:

- Get HRTIM HAL state
- Get captured value
- Get HRTIM timer output level
- Get HRTIM timer output state
- Get delayed protection status
- Get burst status
- Get current push-pull status
- Get idle push-pull status

This section contains the following APIs:

- *HAL_HRTIM_GetState*
- *HAL_HRTIM_GetCapturedValue*
- *HAL_HRTIM_GetCaptured*
- *HAL_HRTIM_GetCapturedDir*

- *HAL_HRTIM_WaveformGetOutputLevel*
- *HAL_HRTIM_WaveformGetOutputState*
- *HAL_HRTIM_GetDelayedProtectionStatus*
- *HAL_HRTIM_GetBurstStatus*
- *HAL_HRTIM_GetCurrentPushPullStatus*
- *HAL_HRTIM_GetIdlePushPullStatus*

28.2.12 Detailed description of functions

HAL_HRTIM_Init

Function name

HAL_StatusTypeDef HAL_HRTIM_Init (HRTIM_HandleTypeDef * hhrtim)

Function description

Initialize a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_DeInit

Function name

HAL_StatusTypeDef HAL_HRTIM_DeInit (HRTIM_HandleTypeDef * hhrtim)

Function description

De-initialize a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_MspInit

Function name

void HAL_HRTIM_MspInit (HRTIM_HandleTypeDef * hhrtim)

Function description

MSP initialization for a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_MspDeInit

Function name

void HAL_HRTIM_MspDeInit (HRTIM_HandleTypeDef * hhrtim)

Function description

MSP de-initialization of a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_TimeBaseConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_TimeBaseConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimeBaseCfgTypeDef * pTimeBaseCfg)

Function description

Configure the time base unit of a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **pTimeBaseCfg:** pointer to the time base configuration structure

Return values

- **HAL:** status

Notes

- This function must be called prior starting the timer
- The time-base unit initialization parameters specify: The timer counter operating mode (continuous, one shot), The timer clock prescaler, The timer period, The timer repetition counter.

HAL_HRTIM_DLLCalibrationStart

Function name

HAL_StatusTypeDef HAL_HRTIM_DLLCalibrationStart (HRTIM_HandleTypeDef * hhrtim, uint32_t CalibrationRate)

Function description

Start the DLL calibration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **CalibrationRate:** DLL calibration period This parameter can be one of the following values:
  - HRTIM_SINGLE_CALIBRATION: One shot DLL calibration
  - HRTIM_CALIBRATIONRATE_0: Periodic DLL calibration. T=6.168 ms
  - HRTIM_CALIBRATIONRATE_1: Periodic DLL calibration. T=0.771 ms
  - HRTIM_CALIBRATIONRATE_2: Periodic DLL calibration. T=0.096 ms
  - HRTIM_CALIBRATIONRATE_3: Periodic DLL calibration. T=0.012 ms

Return values

- **HAL:** status

Notes

- This function locks the HRTIM instance. HRTIM instance is unlocked within the HAL_HRTIM_PollForDLLCalibration function, just before exiting the function.

HAL_HRTIM_DLLCalibrationStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_DLLCalibrationStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t CalibrationRate)

Function description

Start the DLL calibration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **CalibrationRate:** DLL calibration period This parameter can be one of the following values:
  - HRTIM_SINGLE_CALIBRATION: One shot DLL calibration
  - HRTIM_CALIBRATIONRATE_0: Periodic DLL calibration. T=6.168 ms
  - HRTIM_CALIBRATIONRATE_1: Periodic DLL calibration. T=0.771 ms
  - HRTIM_CALIBRATIONRATE_2: Periodic DLL calibration. T=0.096 ms
  - HRTIM_CALIBRATIONRATE_3: Periodic DLL calibration. T=0.012 ms

Return values

- **HAL:** status

Notes

- This function locks the HRTIM instance. HRTIM instance is unlocked within the IRQ processing function when processing the DLL ready interrupt.
- If this function is called for periodic calibration, the DLLRDY interrupt is generated every time the calibration completes which will significantly increases the overall interrupt rate.

HAL_HRTIM_PollForDLLCalibration

Function name

HAL_StatusTypeDef HAL_HRTIM_PollForDLLCalibration (HRTIM_HandleTypeDef * hhrtim, uint32_t Timeout)

Function description

Poll the DLL calibration ready flag and returns when the flag is set (DLL calibration completed) or upon timeout expiration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timeout:** Timeout duration in millisecond

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Start the counter of a timer operating in simple time base mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stop the counter of a timer operating in simple time base mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Start the counter of a timer operating in simple time base mode (Timer repetition interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stop the counter of a timer operating in simple time base mode (Timer repetition interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Start the counter of a timer operating in simple time base mode (Timer repetition DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

HAL_HRTIM_SimpleBaseStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stop the counter of a timer operating in simple time base mode (Timer repetition DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, HRTIM_SimpleOCChannelCfgTypeDef * pSimpleOCChannelCfg)

Function description

Configure an output in simple output compare mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **pSimpleOCChannelCfg:** pointer to the simple output compare output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple output compare mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set according to the selected output compare mode: Toggle: SETxyR = RSTxyR = CMPy Active: SETxyR = CMPy, RSTxyR = 0 Inactive: SETxy =0, RSTxy = CMPy

HAL_HRTIM_SimpleOCStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Start the output compare signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stop the output compare signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Start the output compare signal generation on the designed timer output (Interrupt is enabled (see note note below)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

Notes

- Interrupt enabling depends on the chosen output compare mode Output toggle: compare match interrupt is enabled Output set active: output set interrupt is enabled Output set inactive: output reset interrupt is enabled

HAL_HRTIM_SimpleOCStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stop the output compare signal generation on the designed timer output (Interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Start the output compare signal generation on the designed timer output (DMA request is enabled (see note below)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

Notes

- DMA request enabling depends on the chosen output compare mode Output toggle: compare match DMA request is enabled Output set active: output set DMA request is enabled Output set inactive: output reset DMA request is enabled

HAL_HRTIM_SimpleOCStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stop the output compare signal generation on the designed timer output (DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, HRTIM_SimplePWMChannelCfgTypeDef * pSimplePWMChannelCfg)

Function description

Configure an output in simple PWM mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **pSimplePWMChannelCfg:** pointer to the simple PWM output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple PWM output mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- When Simple PWM mode is used the registers preload mechanism is enabled (otherwise the behavior is not guaranteed).

HAL_HRTIM_SimplePWMStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Start the PWM output signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stop the PWM output signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Start the PWM output signal generation on the designed timer output (The compare interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stop the PWM output signal generation on the designed timer output (The compare interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Start the PWM output signal generation on the designed timer output (The compare DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stop the PWM output signal generation on the designed timer output (The compare DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, HRTIM_SimpleCaptureChannelCfgTypeDef * pSimpleCaptureChannelCfg)

Function description

Configure a simple capture.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Capture unit This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **pSimpleCaptureChannelCfg:** pointer to the simple capture configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple capture mode the capture is trigerred by the designated external event and GPIO input is implicitly used as event source. The cature can be triggered by a rising edge, a falling edge or both edges on event channel.

HAL_HRTIM_SimpleCaptureStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Enable a simple capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

Notes

- The external event triggering the capture is available for all timing units. It can be used directly and is active as soon as the timing unit counter is enabled.

HAL_HRTIM_SimpleCaptureStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Disable a simple capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Enable a simple capture on the designed capture unit (Capture interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Disable a simple capture on the designed capture unit (Capture interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Enable a simple capture on the designed capture unit (Capture DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Disable a simple capture on the designed capture unit (Capture DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel, HRTIM_SimpleOnePulseChannelCfgTypeDef * pSimpleOnePulseChannelCfg)

Function description

Configure an output simple one pulse mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **pSimpleOnePulseChannelCfg:** pointer to the simple one pulse output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple one pulse mode: the timer counter is implicitly started by the reset event, the reset of the timer counter is triggered by the designated external event GPIO input is implicitly used as event source, Output 1 is implicitly controlled by the compare unit 1, Output 2 is implicitly controlled by the compare unit 2. Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- If HAL_HRTIM_SimpleOnePulseChannelConfig is called for both timer outputs, the reset event related configuration data provided in the second call will override the reset event related configuration data provided in the first call.

HAL_HRTIM_SimpleOnePulseStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Enable the simple one pulse signal generation on the designed output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Disable the simple one pulse signal generation on the designed output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Enable the simple one pulse signal generation on the designed output (The compare interrupt is enabled (pulse start)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Disable the simple one pulse signal generation on the designed output (The compare interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_BurstModeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeConfig (HRTIM_HandleTypeDef * hhrtim, HRTIM_BurstModeCfgTypeDef * pBurstModeCfg)

Function description

Configure the burst mode feature of the HRTIM.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **pBurstModeCfg:** pointer to the burst mode configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the burst mode controller

HAL_HRTIM_EventConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_EventConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Event, HRTIM_EventCfgTypeDef * pEventCfg)

Function description

Configure the conditioning of an external event.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Event:** external event to configure This parameter can be one of the following values:
  - HRTIM_EVENT_NONE: no external Event
  - HRTIM_EVENT_1: External event 1
  - HRTIM_EVENT_2: External event 2
  - HRTIM_EVENT_3: External event 3
  - HRTIM_EVENT_4: External event 4
  - HRTIM_EVENT_5: External event 5
  - HRTIM_EVENT_6: External event 6
  - HRTIM_EVENT_7: External event 7
  - HRTIM_EVENT_8: External event 8
  - HRTIM_EVENT_9: External event 9
  - HRTIM_EVENT_10: External event 10
- **pEventCfg:** pointer to the event conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_EventPrescalerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_EventPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Prescaler)

Function description

Configure the external event conditioning block prescaler.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Prescaler:** Prescaler value This parameter can be one of the following values:
  - HRTIM_EVENTPRESCALER_DIV1: fEEVS=fHRTIM
  - HRTIM_EVENTPRESCALER_DIV2: fEEVS=fHRTIM / 2
  - HRTIM_EVENTPRESCALER_DIV4: fEEVS=fHRTIM / 4
  - HRTIM_EVENTPRESCALER_DIV8: fEEVS=fHRTIM / 8

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_FaultConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault, HRTIM_FaultCfgTypeDef * pFaultCfg)

Function description

Configure the conditioning of fault input.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Fault:** fault input to configure This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6
- **pFaultCfg:** pointer to the fault conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer and before enabling faults inputs

HAL_HRTIM_FaultPrescalerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Prescaler)

Function description

Configure the fault conditioning block prescaler.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Prescaler:** Prescaler value This parameter can be one of the following values:
  - HRTIM_FAULTPRESCALER_DIV1: fFLTS=fHRTIM
  - HRTIM_FAULTPRESCALER_DIV2: fFLTS=fHRTIM / 2
  - HRTIM_FAULTPRESCALER_DIV4: fFLTS=fHRTIM / 4
  - HRTIM_FAULTPRESCALER_DIV8: fFLTS=fHRTIM / 8

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer and before enabling faults inputs

HAL_HRTIM_FaultBlankingConfigAndEnable

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultBlankingConfigAndEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault, HRTIM_FaultBlankingCfgTypeDef * pFaultBlkCfg)

Function description

Configure and Enable the blanking source of a Fault input.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Fault:** fault input to configure This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6
- **pFaultBlkCfg:** pointer to the fault conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function automatically enables the Blanking on Fault
- This function must be called when fault is not enabled

HAL_HRTIM_FaultCounterConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultCounterConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault, HRTIM_FaultBlankingCfgTypeDef * pFaultBlkCfg)

Function description

Configure the Fault Counter (Threshold and Reset Mode)

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Fault:** fault input to configure This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6
- **pFaultBlkCfg:** pointer to the fault conditioning configuration structure

Return values

- **HAL:** status
- **HAL:** status

Notes

- A fault is considered valid when the number of events is equal to the (FLTxCNT[3:0]+1) value

HAL_HRTIM_FaultCounterReset

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultCounterReset (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault)

Function description

Reset the fault Counter Reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Fault:** fault input to reset This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6

Return values

- **HAL:** status

HAL_HRTIM_SwapTimerOutput

Function name

HAL_StatusTypeDef HAL_HRTIM_SwapTimerOutput (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Swap the Timer outputs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software register update This parameter can be any combination of the following values:
  - HRTIM_TIMERSWAP_A
  - HRTIM_TIMERSWAP_B
  - HRTIM_TIMERSWAP_C
  - HRTIM_TIMERSWAP_D
  - HRTIM_TIMERSWAP_E
  - HRTIM_TIMERSWAP_F

Return values

- **HAL:** status

Notes

- The function is not significant when the Push-pull mode is enabled (PSHPLL = 1)

HAL_HRTIM_FaultModeCtl

Function name

void HAL_HRTIM_FaultModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Faults, uint32_t Enable)

Function description

Enable or disables the HRTIMx Fault mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Faults:** fault input(s) to enable or disable This parameter can be any combination of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6
- **Enable:** Fault(s) enabling This parameter can be one of the following values:
  - HRTIM_FAULTMODECTL_ENABLED: Fault(s) enabled
  - HRTIM_FAULTMODECTL_DISABLED: Fault(s) disabled

Return values

- **None:**

HAL_HRTIM_ADCTriggerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ADCTriggerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t ADCTrigger, HRTIM_ADCTriggerCfgTypeDef * pADCTriggerCfg)

Function description

Configure both the ADC trigger register update source and the ADC trigger source.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **ADCTrigger:** ADC trigger to configure This parameter can be one of the following values:
  - HRTIM_ADCTRIGGER_1: ADC trigger 1
  - HRTIM_ADCTRIGGER_2: ADC trigger 2
  - HRTIM_ADCTRIGGER_3: ADC trigger 3
  - HRTIM_ADCTRIGGER_4: ADC trigger 4
  - HRTIM_ADCTRIGGER_5: ADC trigger 5
  - HRTIM_ADCTRIGGER_6: ADC trigger 6
  - HRTIM_ADCTRIGGER_7: ADC trigger 7
  - HRTIM_ADCTRIGGER_8: ADC trigger 8
  - HRTIM_ADCTRIGGER_9: ADC trigger 9
  - HRTIM_ADCTRIGGER_10: ADC trigger 10
- **pADCTriggerCfg:** pointer to the ADC trigger configuration structure for Trigger nb (1..4): pADCTriggerCfg->Trigger parameter can be a combination of the following values
  - HRTIM_ADCTRIGGEREVENT13_...
  - HRTIM_ADCTRIGGEREVENT24_... for Trigger nb (5..10): pADCTriggerCfg->Trigger parameter can be one of the following values
  - HRTIM_ADCTRIGGEREVENT579_...
  - HRTIM_ADCTRIGGEREVENT6810_...

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ADCPostScalerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ADCPostScalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t ADCTrigger, uint32_t Postscaler)

Function description

Configure the ADC trigger postscaler register of the ADC trigger source.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **ADCTrigger:** ADC trigger to configure This parameter can be one of the following values:
  - HRTIM_ADCTRIGGER_1: ADC trigger 1
  - HRTIM_ADCTRIGGER_2: ADC trigger 2
  - HRTIM_ADCTRIGGER_3: ADC trigger 3
  - HRTIM_ADCTRIGGER_4: ADC trigger 4
  - HRTIM_ADCTRIGGER_5: ADC trigger 5
  - HRTIM_ADCTRIGGER_6: ADC trigger 6
  - HRTIM_ADCTRIGGER_7: ADC trigger 7
  - HRTIM_ADCTRIGGER_8: ADC trigger 8
  - HRTIM_ADCTRIGGER_9: ADC trigger 9
  - HRTIM_ADCTRIGGER_10: ADC trigger 10
- **Postscaler:** value 0..1F

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_RollOverModeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_RollOverModeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t RollOverCfg)

Function description

Configure the ADC Roll-Over mode of the ADC trigger source.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **RollOverCfg:** This parameter can be a combination of all the following values:
  - HRTIM_TIM_FEROM_BOTH or HRTIM_TIM_FEROM_CREST or HRTIM_TIM_FEROM_VALLEY
  - HRTIM_TIM_BMROM_BOTH or HRTIM_TIM_BMROM_CREST or HRTIM_TIM_BMROM_VALLEY
  - HRTIM_TIM_ADROM_BOTH or HRTIM_TIM_ADROM_CREST or HRTIM_TIM_ADROM_VALLEY
  - HRTIM_TIM_OUTROM_BOTH or HRTIM_TIM_OUTROM_CREST or HRTIM_TIM_OUTROM_VALLEY
  - HRTIM_TIM_ROM_BOTH or HRTIM_TIM_ROM_CREST or HRTIM_TIM_ROM_VALLEY

Notes

- This function must be called before starting the timer

HAL_HRTIM_OutputSwapEnable

Function name

HAL_StatusTypeDef HAL_HRTIM_OutputSwapEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Swap the output of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software register update This parameter can be any combination of the following values:
  - HRTIM_TIMERSWAP_A
  - HRTIM_TIMERSWAP_B
  - HRTIM_TIMERSWAP_C
  - HRTIM_TIMERSWAP_D
  - HRTIM_TIMERSWAP_E
  - HRTIM_TIMERSWAP_F

Return values

- **HAL:** status

HAL_HRTIM_OutputSwapDisable

Function name

HAL_StatusTypeDef HAL_HRTIM_OutputSwapDisable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Un-swap the output of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software register update This parameter can be any combination of the following values:
  - HRTIM_TIMERSWAP_A
  - HRTIM_TIMERSWAP_B
  - HRTIM_TIMERSWAP_C
  - HRTIM_TIMERSWAP_D
  - HRTIM_TIMERSWAP_E
  - HRTIM_TIMERSWAP_F

Return values

- **HAL:** status

HAL_HRTIM_WaveformTimerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformTimerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimerCfgTypeDef * pTimerCfg)

Function description

Configure the general behavior of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **pTimerCfg:** pointer to the timer configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in waveform mode, all the features supported by the HRTIM are available without any limitation.
- This function must be called before starting the timer

HAL_HRTIM_WaveformTimerControl

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformTimerControl (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimerCtlTypeDef * pTimerCtl)

Function description

Configure the general behavior of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **pTimerCtl:** pointer to the timer configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in waveform mode, all the features supported by the HRTIM are available without any limitation.
- This function must be called before starting the timer

HAL_HRTIM_TimerDualChannelDacConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_TimerDualChannelDacConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimerCtlTypeDef * pTimerCtl)

Function description

Configure the Dual Channel Dac behavior of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **pTimerCtl:** pointer to the timer DualChannel Dac configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in waveform mode, all the features supported by the HRTIM are available without any limitation.
- This function must be called before starting the timer

HAL_HRTIM_WaveformCompareConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCompareConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CompareUnit, HRTIM_CompareCfgTypeDef * pCompareCfg)

Function description

Configure the compare unit of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CompareUnit:** Compare unit to configure This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4
- **pCompareCfg:** pointer to the compare unit configuration structure

Return values

- **HAL:** status

Notes

- When auto delayed mode is required for compare unit 2 or compare unit 4, application has to configure separately the capture unit. Capture unit to configure in that case depends on the compare unit auto delayed mode is applied to (see below): Auto delayed on output compare 2: capture unit 1 must be configured Auto delayed on output compare 4: capture unit 2 must be configured
- This function must be called before starting the timer

HAL_HRTIM_WaveformCaptureConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCaptureConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit, HRTIM_CaptureCfgTypeDef * pCaptureCfg)

Function description

Configure the capture unit of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureUnit:** Capture unit to configure This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **pCaptureCfg:** pointer to the compare unit configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformOutputConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, HRTIM_OutputCfgTypeDef * pOutputCfg)

Function description

Configure the output of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **pOutputCfg:** pointer to the timer output configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before configuring the timer and after configuring the deadtime insertion feature (if required).

HAL_HRTIM_WaveformSetOutputLevel

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformSetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, uint32_t OutputLevel)

Function description

Force the timer output to its active or inactive state.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2
- **OutputLevel:** indicates whether the output is forced to its active or inactive level This parameter can be one of the following values:
  - HRTIM_OUTPUTLEVEL_ACTIVE: output is forced to its active level
  - HRTIM_OUTPUTLEVEL_INACTIVE: output is forced to its inactive level

Return values

- **HAL:** status

Notes

- The 'software set/reset trigger' bit in the output set/reset registers is automatically reset by hardware

HAL_HRTIM_TimerEventFilteringConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_TimerEventFilteringConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Event, HRTIM_TimerEventFilteringCfgTypeDef * pTimerEventFilteringCfg)

Function description

Configure the event filtering capabilities of a timer (blanking, windowing)

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **Event:** external event for which timer event filtering must be configured This parameter can be one of the following values:
  - HRTIM_EVENT_1: External event 1
  - HRTIM_EVENT_2: External event 2
  - HRTIM_EVENT_3: External event 3
  - HRTIM_EVENT_4: External event 4
  - HRTIM_EVENT_5: External event 5
  - HRTIM_EVENT_6: External event 6
  - HRTIM_EVENT_7: External event 7
  - HRTIM_EVENT_8: External event 8
  - HRTIM_EVENT_9: External event 9
  - HRTIM_EVENT_10: External event 10
- **pTimerEventFilteringCfg:** pointer to the timer event filtering configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ExtEventCounterConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ExtEventCounterConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t EventCounter, HRTIM_ExternalEventCfgTypeDef * pTimerExternalEventCfg)

Function description

Configure the external Event Counter A or B of a timer (source, threshold, reset mode) but does not enable : call HAL_HRTIM_ExternalEventCounterEnable afterwards.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **EventCounter:** external event Counter A or B for which timer event must be configured This parameter can be one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B
- **pTimerExternalEventCfg:** pointer to the timer external event configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ExtEventCounterEnable

Function name

HAL_StatusTypeDef HAL_HRTIM_ExtEventCounterEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t EventCounter)

Function description

Enable the external event Counter A or B of a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **EventCounter:** external Event Counter A or B for which timer event must be configured This parameter can be a one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ExtEventCounterDisable

Function name

HAL_StatusTypeDef HAL_HRTIM_ExtEventCounterDisable (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t EventCounter)

Function description

Disable the external event Counter A or B of a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **EventCounter:** external event Counter A or B for which timer event must be configured This parameter can be a one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B

Return values

- **HAL:** status

HAL_HRTIM_ExtEventCounterReset

Function name

HAL_StatusTypeDef HAL_HRTIM_ExtEventCounterReset (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t EventCounter)

Function description

Reset the external event Counter A or B of a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **EventCounter:** external event Counter A or B for which timer event must be configured This parameter can be one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_DeadTimeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_DeadTimeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_DeadTimeCfgTypeDef * pDeadTimeCfg)

Function description

Configure the dead-time insertion feature for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **pDeadTimeCfg:** pointer to the deadtime insertion configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ChopperModeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ChopperModeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_ChopperModeCfgTypeDef * pChopperModeCfg)

Function description

Configure the chopper mode feature for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **pChopperModeCfg:** pointer to the chopper mode configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before configuring the timer output(s)

HAL_HRTIM_BurstDMAConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstDMAConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t RegistersToUpdate)

Function description

Configure the burst DMA controller for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **RegistersToUpdate:** registers to be written by DMA This parameter can be any combination of the following values:
  - HRTIM_BURSTDMA_CR: HRTIM_MCR or HRTIM_TIMxCR
  - HRTIM_BURSTDMA_ICR: HRTIM_MICR or HRTIM_TIMxICR
  - HRTIM_BURSTDMA_DIER: HRTIM_MDIER or HRTIM_TIMxDIER
  - HRTIM_BURSTDMA_CNT: HRTIM_MCNT or HRTIM_TIMxCNT
  - HRTIM_BURSTDMA_PER: HRTIM_MPER or HRTIM_TIMxPER
  - HRTIM_BURSTDMA_REP: HRTIM_MREP or HRTIM_TIMxREP
  - HRTIM_BURSTDMA_CMP1: HRTIM_MCMP1 or HRTIM_TIMxCMP1
  - HRTIM_BURSTDMA_CMP2: HRTIM_MCMP2 or HRTIM_TIMxCMP2
  - HRTIM_BURSTDMA_CMP3: HRTIM_MCMP3 or HRTIM_TIMxCMP3
  - HRTIM_BURSTDMA_CMP4: HRTIM_MCMP4 or HRTIM_TIMxCMP4
  - HRTIM_BURSTDMA_DTR: HRTIM_TIMxDTR
  - HRTIM_BURSTDMA_SET1R: HRTIM_TIMxSET1R
  - HRTIM_BURSTDMA_RST1R: HRTIM_TIMxRST1R
  - HRTIM_BURSTDMA_SET2R: HRTIM_TIMxSET2R
  - HRTIM_BURSTDMA_RST2R: HRTIM_TIMxRST2R
  - HRTIM_BURSTDMA_EEFR1: HRTIM_TIMxEEFR1
  - HRTIM_BURSTDMA_EEFR2: HRTIM_TIMxEEFR2
  - HRTIM_BURSTDMA_RSTR: HRTIM_TIMxRSTR
  - HRTIM_BURSTDMA_CHPR: HRTIM_TIMxCHPR
  - HRTIM_BURSTDMA_OUTR: HRTIM_TIMxOUTR
  - HRTIM_BURSTDMA_FLTR: HRTIM_TIMxFLTR

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformCountStart

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStart (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Start the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E
  - HRTIM_TIMERID_TIMER_F

Return values

- **HAL:** status

HAL_HRTIM_WaveformCountStop

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStop (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stop the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_A
  - HRTIM_TIMERID_B
  - HRTIM_TIMERID_C
  - HRTIM_TIMERID_D
  - HRTIM_TIMERID_E
  - HRTIM_TIMERID_F

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled

HAL_HRTIM_WaveformCountStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Start the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_A
  - HRTIM_TIMERID_B
  - HRTIM_TIMERID_C
  - HRTIM_TIMERID_D
  - HRTIM_TIMERID_E
  - HRTIM_TIMERID_F

Return values

- **HAL:** status

Notes

- HRTIM interrupts (e.g. faults interrupts) and interrupts related to the timers to start are enabled within this function. Interrupts to enable are selected through HAL_HRTIM_WaveformTimerConfig function.

HAL_HRTIM_WaveformCountStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stop the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_A
  - HRTIM_TIMERID_B
  - HRTIM_TIMERID_C
  - HRTIM_TIMERID_D
  - HRTIM_TIMERID_E
  - HRTIM_TIMERID_F

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled
- All enabled timer related interrupts are disabled.

HAL_HRTIM_WaveformCountStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Start the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E
  - HRTIM_TIMERID_TIMER_F

Return values

- **HAL:** status

Notes

- This function enables the dma request(s) mentionned in the timer configuration data structure for every timers to start.
- The source memory address, the destination memory address and the size of each DMA transfer are specified at timer configuration time (see HAL_HRTIM_WaveformTimerConfig)

HAL_HRTIM_WaveformCountStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stop the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E
  - HRTIM_TIMERID_TIMER_F

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled
- All enabled timer related DMA requests are disabled.

HAL_HRTIM_WaveformOutputStart

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStart (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStart)

Function description

Enable the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output enabling.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **OutputsToStart:** Timer output(s) to enable This parameter can be any combination of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_WaveformOutputStop

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStop (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStop)

Function description

Disable the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output disabling.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **OutputsToStop:** Timer output(s) to disable This parameter can be any combination of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **HAL:** status

HAL_HRTIM_BurstModeCtl

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Enable)

Function description

Enable or disables the HRTIM burst mode controller.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Enable:** Burst mode controller enabling This parameter can be one of the following values:
  - HRTIM_BURSTMODECTL_ENABLED: Burst mode enabled
  - HRTIM_BURSTMODECTL_DISABLED: Burst mode disabled

Return values

- **HAL:** status

Notes

- This function must be called after starting the timer(s)

HAL_HRTIM_BurstModeSoftwareTrigger

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeSoftwareTrigger (HRTIM_HandleTypeDef * hhrtim)

Function description

Trig the burst mode operation.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_SoftwareCapture

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareCapture (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Trig a software capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

Notes

- The 'software capture' bit in the capure configuration register is automatically reset by hardware

HAL_HRTIM_SoftwareUpdate

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareUpdate (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Trig the update of the registers of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software register update This parameter can be any combination of the following values:
  - HRTIM_TIMERUPDATE_MASTER
  - HRTIM_TIMERUPDATE_A
  - HRTIM_TIMERUPDATE_B
  - HRTIM_TIMERUPDATE_C
  - HRTIM_TIMERUPDATE_D
  - HRTIM_TIMERUPDATE_E
  - HRTIM_TIMERUPDATE_F

Return values

- **HAL:** status

Notes

- The 'software update' bits in the HRTIM conrol register 2 register are automatically reset by hardware

HAL_HRTIM_SoftwareReset

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareReset (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Trig the reset of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software counter reset This parameter can be any combination of the following values:
  - HRTIM_TIMERRESET_MASTER
  - HRTIM_TIMERRESET_TIMER_A
  - HRTIM_TIMERRESET_TIMER_B
  - HRTIM_TIMERRESET_TIMER_C
  - HRTIM_TIMERRESET_TIMER_D
  - HRTIM_TIMERRESET_TIMER_E
  - HRTIM_TIMERRESET_TIMER_F

Return values

- **HAL:** status

Notes

- The 'software reset' bits in the HRTIM conrol register 2 are automatically reset by hardware

HAL_HRTIM_BurstDMATransfer

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstDMATransfer (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t BurstBufferAddress, uint32_t BurstBufferLength)

Function description

Start a burst DMA operation to update HRTIM control registers content.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **BurstBufferAddress:** address of the buffer the HRTIM control registers content will be updated from.
- **BurstBufferLength:** size (in WORDS) of the burst buffer.

Return values

- **HAL:** status

Notes

- The TimerIdx parameter determines the dma channel to be used by the DMA burst controller (see below)
  HRTIM_TIMERINDEX_MASTER: DMA channel 2 is used by the DMA burst controller
  HRTIM_TIMERINDEX_TIMER_A: DMA channel 3 is used by the DMA burst controller
  HRTIM_TIMERINDEX_TIMER_B: DMA channel 4 is used by the DMA burst controller
  HRTIM_TIMERINDEX_TIMER_C: DMA channel 5 is used by the DMA burst controller
  HRTIM_TIMERINDEX_TIMER_D: DMA channel 6 is used by the DMA burst controller
  HRTIM_TIMERINDEX_TIMER_E: DMA channel 7 is used by the DMA burst controller
  HRTIM_TIMERINDEX_TIMER_F: DMA channel 8 is used by the DMA burst controller

HAL_HRTIM_UpdateEnable

Function name

HAL_StatusTypeDef HAL_HRTIM_UpdateEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Enable the transfer from preload to active registers for one or several timing units (including master timer).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer(s) concerned by the register preload enabling command This parameter can be any combination of the following values:
  - HRTIM_TIMERUPDATE_MASTER
  - HRTIM_TIMERUPDATE_A
  - HRTIM_TIMERUPDATE_B
  - HRTIM_TIMERUPDATE_C
  - HRTIM_TIMERUPDATE_D
  - HRTIM_TIMERUPDATE_E
  - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

HAL_HRTIM_UpdateDisable

Function name

HAL_StatusTypeDef HAL_HRTIM_UpdateDisable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Disable the transfer from preload to active registers for one or several timing units (including master timer).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer(s) concerned by the register preload disabling command This parameter can be any combination of the following values:
  - HRTIM_TIMERUPDATE_MASTER
  - HRTIM_TIMERUPDATE_A
  - HRTIM_TIMERUPDATE_B
  - HRTIM_TIMERUPDATE_C
  - HRTIM_TIMERUPDATE_D
  - HRTIM_TIMERUPDATE_E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **HAL:** status

HAL_HRTIM_GetState

Function name

HAL_HRTIM_StateTypeDef HAL_HRTIM_GetState (HRTIM_HandleTypeDef * hhrtim)

Function description

Return the HRTIM HAL state.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** state

HAL_HRTIM_GetCapturedValue

Function name

uint32_t HAL_HRTIM_GetCapturedValue (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Return actual value of the capture register of the designated capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **Captured:** value

HAL_HRTIM_GetCapturedDir

Function name

uint32_t HAL_HRTIM_GetCapturedDir (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Return actual direction of the capture register of the designated capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **captured:** direction
  - This parameter is one HRTIM_Timer_UpDown_Mode :
  - HRTIM_TIMERUPDOWNMODE_UP
  - HRTIM_TIMERUPDOWNMODE_UPDOWN

HAL_HRTIM_GetCaptured

Function name

HRTIM_CaptureValueTypeDef HAL_HRTIM_GetCaptured (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Return actual value and direction of the capture register of the designated capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **captured:** value and direction structure

HAL_HRTIM_WaveformGetOutputLevel

Function name

uint32_t HAL_HRTIM_WaveformGetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)

Function description

Return actual level (active or inactive) of the designated output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **Output:** level

Notes

- Returned output level is taken before the output stage (chopper, polarity).

HAL_HRTIM_WaveformGetOutputState

Function name

uint32_t HAL_HRTIM_WaveformGetOutputState (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)

Function description

Return actual state (RUN, IDLE, FAULT) of the designated output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **Output:** state

HAL_HRTIM_GetDelayedProtectionStatus

Function name

uint32_t HAL_HRTIM_GetDelayedProtectionStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)

Function description

Return the level (active or inactive) of the designated output when the delayed protection was triggered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
  - HRTIM_OUTPUT_TF1: Timer F - Output 1
  - HRTIM_OUTPUT_TF2: Timer F - Output 2

Return values

- **Delayed:** protection status

HAL_HRTIM_GetBurstStatus

Function name

uint32_t HAL_HRTIM_GetBurstStatus (HRTIM_HandleTypeDef * hhrtim)

Function description

Return the actual status (active or inactive) of the burst mode controller.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **Burst:** mode controller status

HAL_HRTIM_GetCurrentPushPullStatus

Function name

uint32_t HAL_HRTIM_GetCurrentPushPullStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Indicate on which output the signal is currently active (when the push pull mode is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **Burst:** mode controller status

HAL_HRTIM_GetIdlePushPullStatus

Function name

uint32_t HAL_HRTIM_GetIdlePushPullStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Indicate on which output the signal was applied, in push-pull mode, balanced fault mode or delayed idle mode, when the protection was triggered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **Idle:** Push Pull Status

HAL_HRTIM_IRQHandler

Function name

void HAL_HRTIM_IRQHandler (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

This function handles HRTIM interrupt request.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be any value of HRTIM_Timer_Index

Return values

- **None:**

HAL_HRTIM_Fault1Callback

Function name

void HAL_HRTIM_Fault1Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 1 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle *

Return values

- **None:**
- **None:**

HAL_HRTIM_Fault2Callback

Function name

void HAL_HRTIM_Fault2Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 2 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault3Callback

Function name

void HAL_HRTIM_Fault3Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 3 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault4Callback

Function name

void HAL_HRTIM_Fault4Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 4 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault5Callback

Function name

void HAL_HRTIM_Fault5Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 5 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault6Callback

Function name

void HAL_HRTIM_Fault6Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 6 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_SystemFaultCallback

Function name

void HAL_HRTIM_SystemFaultCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a system fault interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_DLLCalibrationReadyCallback

Function name

void HAL_HRTIM_DLLCalibrationReadyCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when the DLL calibration is completed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_BurstModePeriodCallback

Function name

void HAL_HRTIM_BurstModePeriodCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when the end of the burst mode period is reached.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_SynchronizationEventCallback

Function name

void HAL_HRTIM_SynchronizationEventCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a synchronization input event is received.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_RegistersUpdateCallback

Function name

void HAL_HRTIM_RegistersUpdateCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when timer registers are updated.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_RepetitionEventCallback

Function name

void HAL_HRTIM_RepetitionEventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when timer repetition period has elapsed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Compare1EventCallback

Function name

void HAL_HRTIM_Compare1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 1 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Compare2EventCallback

Function name

void HAL_HRTIM_Compare2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 2 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Compare3EventCallback

Function name

void HAL_HRTIM_Compare3EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 3 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Compare4EventCallback

Function name

void HAL_HRTIM_Compare4EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 4 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Capture1EventCallback

Function name

void HAL_HRTIM_Capture1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x capture 1 event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Capture2EventCallback

Function name

void HAL_HRTIM_Capture2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x capture 2 event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_DelayedProtectionCallback

Function name

void HAL_HRTIM_DelayedProtectionCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the delayed idle or balanced idle mode is entered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_CounterResetCallback

Function name

void HAL_HRTIM_CounterResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x counter reset/roll-over event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Output1SetCallback

Function name

void HAL_HRTIM_Output1SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 1 is set.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Output1ResetCallback

Function name

void HAL_HRTIM_Output1ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 1 is reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Output2SetCallback

Function name

void HAL_HRTIM_Output2SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 2 is set.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_Output2ResetCallback

Function name

void HAL_HRTIM_Output2ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 2 is reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_BurstDMATransferCallback

Function name

void HAL_HRTIM_BurstDMATransferCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when a DMA burst transfer is completed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return values

- **None:**

HAL_HRTIM_ErrorCallback

Function name

void HAL_HRTIM_ErrorCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a DMA error occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

28.3 HRTIM Firmware driver defines

The following section lists the various define and macros of the module.

28.3.1 HRTIM

HRTIM

HRTIM ADC Extended Trigger

HRTIM_ADCTRIGGER_5

ADC trigger 5 identifier

HRTIM_ADCTRIGGER_6

ADC trigger 6 identifier

HRTIM_ADCTRIGGER_7

ADC trigger 7 identifier

HRTIM_ADCTRIGGER_8

ADC trigger 8 identifier

HRTIM_ADCTRIGGER_9

ADC trigger 9 identifier

HRTIM_ADCTRIGGER_10

ADC trigger 10 identifier

IS_HRTIM_ADCTRIGGER

IS_HRTIM_ADCEXTTRIGGER

HRTIM ADC TRIGGER SELECTION

LL_HRTIM_ADCTRIG_SRC6810_MCMP1

ADC extended Trigger on Master Compare 1

LL_HRTIM_ADCTRIG_SRC6810_MCMP2

ADC extended Trigger on Master Compare 2

LL_HRTIM_ADCTRIG_SRC6810_MCMP3

ADC extended Trigger on Master Compare 3

LL_HRTIM_ADCTRIG_SRC6810_MCMP4

ADC extended Trigger on Master Compare 4

LL_HRTIM_ADCTRIG_SRC6810_MPER

ADC extended Trigger on Master Period

LL_HRTIM_ADCTRIG_SRC6810_EEV6

ADC extended Trigger on External Event 6

LL_HRTIM_ADCTRIG_SRC6810_EEV7

ADC extended Trigger on External Event 7

LL_HRTIM_ADCTRIG_SRC6810_EEV8

ADC extended Trigger on External Event 8

LL_HRTIM_ADCTRIG_SRC6810_EEV9

ADC extended Trigger on External Event 9

LL_HRTIM_ADCTRIG_SRC6810_EEV10

ADC extended Trigger on External Event 10

LL_HRTIM_ADCTRIG_SRC6810_TIMA_CMP2

ADC extended Trigger on Timer A Compare 2

LL_HRTIM_ADCTRIG_SRC6810_TIMA_CMP4

ADC extended Trigger on Timer A Compare 4

LL_HRTIM_ADCTRIG_SRC6810_TIMA_PER

ADC extended Trigger on Timer A Period

LL_HRTIM_ADCTRIG_SRC6810_TIMB_CMP2

ADC extended Trigger on Timer B Compare 2

LL_HRTIM_ADCTRIG_SRC6810_TIMB_CMP4

ADC extended Trigger on Timer B Compare 4

LL_HRTIM_ADCTRIG_SRC6810_TIMB_PER

ADC extended Trigger on Timer B Period

LL_HRTIM_ADCTRIG_SRC6810_TIMC_CMP2

ADC extended Trigger on Timer C Compare 2

LL_HRTIM_ADCTRIG_SRC6810_TIMC_CMP4

ADC extended Trigger on Timer C Compare 4

LL_HRTIM_ADCTRIG_SRC6810_TIMC_PER

ADC extended Trigger on Timer C Period

LL_HRTIM_ADCTRIG_SRC6810_TIMC_RST

ADC extended Trigger on Timer C Reset and counter roll-over

LL_HRTIM_ADCTRIG_SRC6810_TIMD_CMP2

ADC extended Trigger on Timer D Compare 2

LL_HRTIM_ADCTRIG_SRC6810_TIMD_CMP4

ADC extended Trigger on Timer D Compare 4

LL_HRTIM_ADCTRIG_SRC6810_TIMD_PER

ADC extended Trigger on Timer D Period

LL_HRTIM_ADCTRIG_SRC6810_TIMD_RST

ADC extended Trigger on Timer D Reset and counter roll-over

LL_HRTIM_ADCTRIG_SRC6810_TIME_CMP2

ADC extended Trigger on Timer E Compare 2

LL_HRTIM_ADCTRIG_SRC6810_TIME_CMP3

ADC extended Trigger on Timer E Compare 3

LL_HRTIM_ADCTRIG_SRC6810_TIME_CMP4

ADC extended Trigger on Timer E Compare 4

LL_HRTIM_ADCTRIG_SRC6810_TIME_RST

ADC extended Trigger on Timer E Reset and counter roll-over

LL_HRTIM_ADCTRIG_SRC6810_TIMF_CMP2

ADC extended Trigger on Timer F Compare 2

LL_HRTIM_ADCTRIG_SRC6810_TIMF_CMP3

ADC extended Trigger on Timer F Compare 3

LL_HRTIM_ADCTRIG_SRC6810_TIMF_CMP4

ADC extended Trigger on Timer F Compare 4

LL_HRTIM_ADCTRIG_SRC6810_TIMF_PER

ADC extended Trigger on Timer F Period

LL_HRTIM_ADCTRIG_SRC579_MCMP1

ADC extended Trigger on Master Compare 1

LL_HRTIM_ADCTRIG_SRC579_MCMP2

ADC extended Trigger on Master Compare 2

LL_HRTIM_ADCTRIG_SRC579_MCMP3

ADC extended Trigger on Master Compare 3

LL_HRTIM_ADCTRIG_SRC579_MCMP4

ADC extended Trigger on Master Compare 4

LL_HRTIM_ADCTRIG_SRC579_MPER

ADC extended Trigger on Master Period

LL_HRTIM_ADCTRIG_SRC579_EEV1

ADC extended Trigger on External Event 1

LL_HRTIM_ADCTRIG_SRC579_EEV2

ADC extended Trigger on External Event 2

LL_HRTIM_ADCTRIG_SRC579_EEV3

ADC extended Trigger on External Event 3

LL_HRTIM_ADCTRIG_SRC579_EEV4

ADC extended Trigger on External Event 4

LL_HRTIM_ADCTRIG_SRC579_EEV5

ADC extended Trigger on External Event 5

LL_HRTIM_ADCTRIG_SRC579_TIMA_CMP3

ADC extended Trigger on Timer A Compare 3

LL_HRTIM_ADCTRIG_SRC579_TIMA_CMP4

ADC extended Trigger on Timer A Compare 4

LL_HRTIM_ADCTRIG_SRC579_TIMA_PER

ADC extended Trigger on Timer A Period

LL_HRTIM_ADCTRIG_SRC579_TIMA_RST

ADC extended Trigger on Timer A Period

LL_HRTIM_ADCTRIG_SRC579_TIMB_CMP3

ADC extended Trigger on Timer B Compare 3

LL_HRTIM_ADCTRIG_SRC579_TIMB_CMP4

ADC extended Trigger on Timer B Compare 4

LL_HRTIM_ADCTRIG_SRC579_TIMB_PER

ADC extended Trigger on Timer B Period

LL_HRTIM_ADCTRIG_SRC579_TIMB_RST

ADC extended Trigger on Timer B Reset and counter roll-over

LL_HRTIM_ADCTRIG_SRC579_TIMC_CMP3

ADC extended Trigger on Timer C Compare 3

LL_HRTIM_ADCTRIG_SRC579_TIMC_CMP4

ADC extended Trigger on Timer C Compare 4

LL_HRTIM_ADCTRIG_SRC579_TIMC_PER

ADC extended Trigger on Timer C Period

LL_HRTIM_ADCTRIG_SRC579_TIMD_CMP3

ADC extended Trigger on Timer D Compare 3

LL_HRTIM_ADCTRIG_SRC579_TIMD_CMP4

ADC extended Trigger on Timer D Compare 4

LL_HRTIM_ADCTRIG_SRC579_TIMD_PER

ADC extended Trigger on Timer D Period

LL_HRTIM_ADCTRIG_SRC579_TIME_CMP3

ADC extended Trigger on Timer E Compare 3

LL_HRTIM_ADCTRIG_SRC579_TIME_CMP4

ADC extended Trigger on Timer E Compare 4

LL_HRTIM_ADCTRIG_SRC579_TIME_PER

ADC extended Trigger on Timer E Period

LL_HRTIM_ADCTRIG_SRC579_TIMF_CMP2

ADC extended Trigger on Timer F Compare 2

LL_HRTIM_ADCTRIG_SRC579_TIMF_CMP3

ADC extended Trigger on Timer F Compare 3

LL_HRTIM_ADCTRIG_SRC579_TIMF_CMP4

ADC extended Trigger on Timer F Compare 4

LL_HRTIM_ADCTRIG_SRC579_TIMF_PER

ADC extended Trigger on Timer F Period

LL_HRTIM_ADCTRIG_SRC579_TIMF_RST

ADC extended Trigger on Timer F Reset and counter roll-over

HRTIM ADC Trigger

HRTIM_ADCTRIGGER_1

ADC trigger 1 identifier

HRTIM_ADCTRIGGER_2

ADC trigger 2 identifier

HRTIM_ADCTRIGGER_3

ADC trigger 3 identifier

HRTIM_ADCTRIGGER_4

ADC trigger 4 identifier

HRTIM ADC Trigger Event

HRTIM_ADCTRIGGEREVENT13_NONE

No ADC trigger event

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP1

ADC Trigger on master compare 1U

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP2

ADC Trigger on master compare 2U

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP3

ADC Trigger on master compare 3U

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP4

ADC Trigger on master compare 4U

HRTIM_ADCTRIGGEREVENT13_MASTER_PERIOD

ADC Trigger on master period

HRTIM_ADCTRIGGEREVENT13_EVENT_1

ADC Trigger on external event 1U

HRTIM_ADCTRIGGEREVENT13_EVENT_2

ADC Trigger on external event 2U

HRTIM_ADCTRIGGEREVENT13_EVENT_3

ADC Trigger on external event 3U

HRTIM_ADCTRIGGEREVENT13_EVENT_4

ADC Trigger on external event 4U

HRTIM_ADCTRIGGEREVENT13_EVENT_5

ADC Trigger on external event 5U

HRTIM_ADCTRIGGEREVENT13_TIMERF_CMP2

ADC Trigger on Timer F compare 2U

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP3

ADC Trigger on Timer A compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP4

ADC Trigger on Timer A compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERA_PERIOD

ADC Trigger on Timer A period

HRTIM_ADCTRIGGEREVENT13_TIMERA_RESET

ADC Trigger on Timer A reset

HRTIM_ADCTRIGGEREVENT13_TIMERF_CMP3

ADC Trigger on Timer F compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP3

ADC Trigger on Timer B compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP4

ADC Trigger on Timer B compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERB_PERIOD

ADC Trigger on Timer B period

HRTIM_ADCTRIGGEREVENT13_TIMERB_RESET

ADC Trigger on Timer B reset

HRTIM_ADCTRIGGEREVENT13_TIMERF_CMP4

ADC Trigger on Timer F compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP3

ADC Trigger on Timer C compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP4

ADC Trigger on Timer C compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERC_PERIOD

ADC Trigger on Timer C period

HRTIM_ADCTRIGGEREVENT13_TIMERF_PERIOD

ADC Trigger on Timer F period

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP3

ADC Trigger on Timer D compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP4

ADC Trigger on Timer D compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERD_PERIOD

ADC Trigger on Timer D period

HRTIM_ADCTRIGGEREVENT13_TIMERF_RESET

ADC Trigger on Timer F reset

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP3

ADC Trigger on Timer E compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP4

ADC Trigger on Timer E compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERE_PERIOD

ADC Trigger on Timer E period

HRTIM_ADCTRIGGEREVENT24_NONE

No ADC trigger event

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP1

ADC Trigger on master compare 1U

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP2

ADC Trigger on master compare 2U

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP3

ADC Trigger on master compare 3U

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP4

ADC Trigger on master compare 4U

HRTIM_ADCTRIGGEREVENT24_MASTER_PERIOD

ADC Trigger on master period

HRTIM_ADCTRIGGEREVENT24_EVENT_6

ADC Trigger on external event 6U

HRTIM_ADCTRIGGEREVENT24_EVENT_7

ADC Trigger on external event 7U

HRTIM_ADCTRIGGEREVENT24_EVENT_8

ADC Trigger on external event 8U

HRTIM_ADCTRIGGEREVENT24_EVENT_9

ADC Trigger on external event 9U

HRTIM_ADCTRIGGEREVENT24_EVENT_10

ADC Trigger on external event 10U

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP2

ADC Trigger on Timer A compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERF_CMP2

ADC Trigger on Timer F compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP4

ADC Trigger on Timer A compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERA_PERIOD

ADC Trigger on Timer A period

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP2

ADC Trigger on Timer B compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERF_CMP3

ADC Trigger on Timer F compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP4

ADC Trigger on Timer B compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERB_PERIOD

ADC Trigger on Timer B period

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP2

ADC Trigger on Timer C compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERF_CMP4

ADC Trigger on Timer F compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP4

ADC Trigger on Timer C compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERC_PERIOD

ADC Trigger on Timer C period

HRTIM_ADCTRIGGEREVENT24_TIMERC_RESET

ADC Trigger on Timer C reset

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP2

ADC Trigger on Timer D compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERF_PERIOD

ADC Trigger on Timer F period

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP4

ADC Trigger on Timer D compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERD_PERIOD

ADC Trigger on Timer D period

HRTIM_ADCTRIGGEREVENT24_TIMERD_RESET

ADC Trigger on Timer D reset

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP2

ADC Trigger on Timer E compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP3

ADC Trigger on Timer E compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP4

ADC Trigger on Timer E compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERE_RESET

ADC Trigger on Timer E reset

HRTIM_ADCTRIGGEREVENT6810_MASTER_CMP1

ADC Trigger on master compare 1U

HRTIM_ADCTRIGGEREVENT6810_MASTER_CMP2

ADC Trigger on master compare 2U

HRTIM_ADCTRIGGEREVENT6810_MASTER_CMP3

ADC Trigger on master compare 3U

HRTIM_ADCTRIGGEREVENT6810_MASTER_CMP4

ADC Trigger on master compare 4U

HRTIM_ADCTRIGGEREVENT6810_MASTER_PERIOD

ADC Trigger on master period

HRTIM_ADCTRIGGEREVENT6810_EVENT_6

ADC Trigger on external event 6U

HRTIM_ADCTRIGGEREVENT6810_EVENT_7

ADC Trigger on external event 7U

HRTIM_ADCTRIGGEREVENT6810_EVENT_8

ADC Trigger on external event 8U

HRTIM_ADCTRIGGEREVENT6810_EVENT_9

ADC Trigger on external event 9U

HRTIM_ADCTRIGGEREVENT6810_EVENT_10

ADC Trigger on external event 10U

HRTIM_ADCTRIGGEREVENT6810_TIMERA_CMP2

ADC Trigger on Timer A compare 2U

HRTIM_ADCTRIGGEREVENT6810_TIMERA_CMP4

ADC Trigger on Timer A compare 4U

HRTIM_ADCTRIGGEREVENT6810_TIMERA_PERIOD

ADC Trigger on Timer A period

HRTIM_ADCTRIGGEREVENT6810_TIMERB_CMP2

ADC Trigger on Timer B compare 2U

HRTIM_ADCTRIGGEREVENT6810_TIMERB_CMP4

ADC Trigger on Timer B compare 4U

HRTIM_ADCTRIGGEREVENT6810_TIMERB_PERIOD

ADC Trigger on Timer B period

HRTIM_ADCTRIGGEREVENT6810_TIMERC_CMP2

ADC Trigger on Timer C compare 2U

HRTIM_ADCTRIGGEREVENT6810_TIMERC_CMP4

ADC Trigger on Timer C compare 4U

HRTIM_ADCTRIGGEREVENT6810_TIMERC_PERIOD

ADC Trigger on Timer C period

HRTIM_ADCTRIGGEREVENT6810_TIMERC_RESET

ADC Trigger on Timer C reset

HRTIM_ADCTRIGGEREVENT6810_TIMERD_CMP2

ADC Trigger on Timer D compare 2U

HRTIM_ADCTRIGGEREVENT6810_TIMERD_CMP4

ADC Trigger on Timer D compare 4U

HRTIM_ADCTRIGGEREVENT6810_TIMERD_PERIOD

ADC Trigger on Timer D period

HRTIM_ADCTRIGGEREVENT6810_TIMERD_RESET

ADC Trigger on Timer D reset

HRTIM_ADCTRIGGEREVENT6810_TIMERE_CMP2

ADC Trigger on Timer E compare 2U

HRTIM_ADCTRIGGEREVENT6810_TIMERE_CMP3

ADC Trigger on Timer E compare 3U

HRTIM_ADCTRIGGEREVENT6810_TIMERE_CMP4

ADC Trigger on Timer E compare 4U

HRTIM_ADCTRIGGEREVENT6810_TIMERE_RESET

ADC Trigger on Timer E reset

HRTIM_ADCTRIGGEREVENT6810_TIMERF_CMP2

ADC Trigger on Timer F compare 2U

HRTIM_ADCTRIGGEREVENT6810_TIMERF_CMP3

ADC Trigger on Timer F compare 3U

HRTIM_ADCTRIGGEREVENT6810_TIMERF_CMP4

ADC Trigger on Timer F compare 4U

HRTIM_ADCTRIGGEREVENT6810_TIMERF_PERIOD

ADC Trigger on Timer F period

HRTIM_ADCTRIGGEREVENT579_MASTER_CMP1

ADC Trigger on master compare 1U

HRTIM_ADCTRIGGEREVENT579_MASTER_CMP2

ADC Trigger on master compare 2U

HRTIM_ADCTRIGGEREVENT579_MASTER_CMP3

ADC Trigger on master compare 3U

HRTIM_ADCTRIGGEREVENT579_MASTER_CMP4

ADC Trigger on master compare 4U

HRTIM_ADCTRIGGEREVENT579_MASTER_PERIOD

ADC Trigger on master period

HRTIM_ADCTRIGGEREVENT579_EVENT_1

ADC Trigger on external event 1U

HRTIM_ADCTRIGGEREVENT579_EVENT_2

ADC Trigger on external event 2U

HRTIM_ADCTRIGGEREVENT579_EVENT_3

ADC Trigger on external event 3U

HRTIM_ADCTRIGGEREVENT579_EVENT_4

ADC Trigger on external event 4U

HRTIM_ADCTRIGGEREVENT579_EVENT_5

ADC Trigger on external event 5U

HRTIM_ADCTRIGGEREVENT579_TIMERA_CMP3

ADC Trigger on Timer A compare 3U

HRTIM_ADCTRIGGEREVENT579_TIMERA_CMP4

ADC Trigger on Timer A compare 4U

HRTIM_ADCTRIGGEREVENT579_TIMERA_PERIOD

ADC Trigger on Timer A period

HRTIM_ADCTRIGGEREVENT579_TIMERA_RESET

ADC Trigger on Timer A reset

HRTIM_ADCTRIGGEREVENT579_TIMERB_CMP3

ADC Trigger on Timer B compare 3U

HRTIM_ADCTRIGGEREVENT579_TIMERB_CMP4

ADC Trigger on Timer B compare 4U

HRTIM_ADCTRIGGEREVENT579_TIMERB_PERIOD

ADC Trigger on Timer B period

HRTIM_ADCTRIGGEREVENT579_TIMERB_RESET

ADC Trigger on Timer B reset

HRTIM_ADCTRIGGEREVENT579_TIMERC_CMP3

ADC Trigger on Timer C compare 3U

HRTIM_ADCTRIGGEREVENT579_TIMERC_CMP4

ADC Trigger on Timer C compare 4U

HRTIM_ADCTRIGGEREVENT579_TIMERC_PERIOD

ADC Trigger on Timer C period

HRTIM_ADCTRIGGEREVENT579_TIMERD_CMP3

ADC Trigger on Timer D compare 3U

HRTIM_ADCTRIGGEREVENT579_TIMERD_CMP4

ADC Trigger on Timer D compare 4U

HRTIM_ADCTRIGGEREVENT579_TIMERD_PERIOD

ADC Trigger on Timer D period

HRTIM_ADCTRIGGEREVENT579_TIMERE_CMP3

ADC Trigger on Timer E compare 3U

HRTIM_ADCTRIGGEREVENT579_TIMERE_CMP4

ADC Trigger on Timer E compare 4U

HRTIM_ADCTRIGGEREVENT579_TIMERE_PERIOD

ADC Trigger on Timer E period

HRTIM_ADCTRIGGEREVENT579_TIMERF_CMP2

ADC Trigger on Timer F compare 2U

HRTIM_ADCTRIGGEREVENT579_TIMERF_CMP3

ADC Trigger on Timer F compare 3U

HRTIM_ADCTRIGGEREVENT579_TIMERF_CMP4

ADC Trigger on Timer F compare 4U

HRTIM_ADCTRIGGEREVENT579_TIMERF_PERIOD

ADC Trigger on Timer F period

HRTIM_ADCTRIGGEREVENT579_TIMERF_RESET

ADC Trigger on Timer F reset

HRTIM ADC Trigger Update Source

HRTIM_ADCTRIGGERUPDATE_MASTER

Master timer

HRTIM_ADCTRIGGERUPDATE_TIMER_A

Timer A

HRTIM_ADCTRIGGERUPDATE_TIMER_B

Timer B

HRTIM_ADCTRIGGERUPDATE_TIMER_C

Timer C

HRTIM_ADCTRIGGERUPDATE_TIMER_D

Timer D

HRTIM_ADCTRIGGERUPDATE_TIMER_E

Timer E

HRTIM_ADCTRIGGERUPDATE_TIMER_F

Timer F

HRTIM Burst DMA Registers Update

HRTIM_BURSTDMA_NONE

No register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CR

MCR or TIMxCR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_ICR

MICR or TIMxICR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_DIER

MDIER or TIMxDIER register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CNT

MCNTR or CNTxCR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_PER

MPER or PERxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_REP

MREPR or REPxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP1

MCMP1R or CMP1xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP2

MCMP2R or CMP2xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP3

MCMP3R or CMP3xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP4

MCMP4R or CMP4xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_DTR

TDxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_SET1R

SET1R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_RST1R

RST1R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_SET2R

SET2R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_RST2R

RST1R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_EEFR1

EEFxR1 register is updated by Burst DMA accesses

HRTIM_BURSTDMA_EEFR2

EEFxR2 register is updated by Burst DMA accesses

HRTIM_BURSTDMA_RSTR

RSTxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CHPR

CHPxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_OUTR

OUTxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_FLTR

FLTxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CR2

TIMxCR2 register is updated by Burst DMA accesses

HRTIM_BURSTDMA_EEFR3

EEFxR3 register is updated by Burst DMA accesses

HRTIM Burst Mode Clock Source

HRTIM_BURSTMODECLOCKSOURCE_MASTER

Master timer counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_A

Timer A counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_B

Timer B counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_C

Timer C counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_D

Timer D counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_E

Timer E counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_F

Timer F counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIM16_OC

On-chip Event 1 (BMClk[1]), acting as a burst mode counter clock

HRTIM_BURSTMODECLOCKSOURCE_TIM17_OC

On-chip Event 2 (BMClk[2]), acting as a burst mode counter clock

HRTIM_BURSTMODECLOCKSOURCE_TIM7_TRGO

On-chip Event 3 (BMClk[3]), acting as a burst mode counter clock

HRTIM_BURSTMODECLOCKSOURCE_FHRTIM

Prescaled fHRTIM clock is used as clock source for the burst mode counter

HRTIM Burst Mode Control

HRTIM_BURSTMODECTL_DISABLED

Burst mode disabled

HRTIM_BURSTMODECTL_ENABLED

Burst mode enabled

HRTIM Burst Mode Operating Mode

HRTIM_BURSTMODE_SINGLESHOT

Burst mode operates in single shot mode

HRTIM_BURSTMODE_CONTINOUS

Burst mode operates in continuous mode

HRTIM Burst Mode Prescaler

HRTIM_BURSTMODEPRESCALER_DIV1

fBRST = fHRTIM

HRTIM_BURSTMODEPRESCALER_DIV2

fBRST = fHRTIM/2U

HRTIM_BURSTMODEPRESCALER_DIV4

fBRST = fHRTIM/4U

HRTIM_BURSTMODEPRESCALER_DIV8

fBRST = fHRTIM/8U

HRTIM_BURSTMODEPRESCALER_DIV16

fBRST = fHRTIM/16U

HRTIM_BURSTMODEPRESCALER_DIV32

fBRST = fHRTIM/32U

HRTIM_BURSTMODEPRESCALER_DIV64

fBRST = fHRTIM/64U

HRTIM_BURSTMODEPRESCALER_DIV128

fBRST = fHRTIM/128U

HRTIM_BURSTMODEPRESCALER_DIV256

fBRST = fHRTIM/256U

HRTIM_BURSTMODEPRESCALER_DIV512

fBRST = fHRTIM/512U

HRTIM_BURSTMODEPRESCALER_DIV1024

fBRST = fHRTIM/1024U

HRTIM_BURSTMODEPRESCALER_DIV2048

fBRST = fHRTIM/2048U

HRTIM_BURSTMODEPRESCALER_DIV4096

fBRST = fHRTIM/4096U

HRTIM_BURSTMODEPRESCALER_DIV8192

fBRST = fHRTIM/8192U

HRTIM_BURSTMODEPRESCALER_DIV16384

fBRST = fHRTIM/16384U

HRTIM_BURSTMODEPRESCALER_DIV32768

fBRST = fHRTIM/32768U

HRTIM Burst Mode Register Preload Enable

HRIM_BURSTMODEPRELOAD_DISABLED

Preload disabled: the write access is directly done into active registers

HRIM_BURSTMODEPRELOAD_ENABLED

Preload enabled: the write access is done into preload registers

HRTIM Burst Mode Status

HRTIM_BURSTMODESTATUS_NORMAL

Normal operation

HRTIM_BURSTMODESTATUS_ONGOING

Burst operation on-going

HRTIM Burst Mode Trigger

HRTIM_BURSTMODETRIGGER_NONE

No trigger

HRTIM_BURSTMODETRIGGER_MASTER_RESET

Master reset

HRTIM_BURSTMODETRIGGER_MASTER_REPETITION

Master repetition

HRTIM_BURSTMODETRIGGER_MASTER_CMP1

Master compare 1U

HRTIM_BURSTMODETRIGGER_MASTER_CMP2

Master compare 2U

HRTIM_BURSTMODETRIGGER_MASTER_CMP3

Master compare 3U

HRTIM_BURSTMODETRIGGER_MASTER_CMP4

Master compare 4U

HRTIM_BURSTMODETRIGGER_TIMERA_RESET

Timer A reset

HRTIM_BURSTMODETRIGGER_TIMERA_REPETITION

Timer A repetition

HRTIM_BURSTMODETRIGGER_TIMERA_CMP1

Timer A compare 1

HRTIM_BURSTMODETRIGGER_TIMERA_CMP2

Timer A compare 2

HRTIM_BURSTMODETRIGGER_TIMERB_RESET

Timer B reset

HRTIM_BURSTMODETRIGGER_TIMERB_REPETITION

Timer B repetition

HRTIM_BURSTMODETRIGGER_TIMERB_CMP1

Timer B compare 1

HRTIM_BURSTMODETRIGGER_TIMERB_CMP2

Timer B compare 2

HRTIM_BURSTMODETRIGGER_TIMERC_RESET

Timer C reset

HRTIM_BURSTMODETRIGGER_TIMERC_REPETITION

Timer C repetition

HRTIM_BURSTMODETRIGGER_TIMERC_CMP1

Timer C compare 1

HRTIM_BURSTMODETRIGGER_TIMERF_RESET

Timer F reset

HRTIM_BURSTMODETRIGGER_TIMERD_RESET

Timer D reset

HRTIM_BURSTMODETRIGGER_TIMERD_REPETITION

Timer D repetition

HRTIM_BURSTMODETRIGGER_TIMERF_REPETITION

Timer F repetition

HRTIM_BURSTMODETRIGGER_TIMERD_CMP2

Timer D compare 2

HRTIM_BURSTMODETRIGGER_TIMERF_CMP1

Timer F compare 1

HRTIM_BURSTMODETRIGGER_TIMERE_REPETITION

Timer E repetition

HRTIM_BURSTMODETRIGGER_TIMERE_CMP1

Timer E compare 1

HRTIM_BURSTMODETRIGGER_TIMERE_CMP2

Timer E compare 2

HRTIM_BURSTMODETRIGGER_TIMERA_EVENT7

Timer A period following External Event 7

HRTIM_BURSTMODETRIGGER_TIMERD_EVENT8

Timer D period following External Event 8

HRTIM_BURSTMODETRIGGER_EVENT_7

External Event 7 (timer A filters applied)

HRTIM_BURSTMODETRIGGER_EVENT_8

External Event 8 (timer D filters applied)

HRTIM_BURSTMODETRIGGER_EVENT_ONCHIP

On-chip Event

HRTIM Capture Unit

HRTIM_CAPTUREUNIT_1

Capture unit 1 identifier

HRTIM_CAPTUREUNIT_2

Capture unit 2 identifier

HRTIM Capture Unit TimerF Trigger

HRTIM_CAPTURETRIGGER_TF1_SET

Capture is triggered by TF1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TF1_RESET

Capture is triggered by TF1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERF_CMP1

Timer F Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERF_CMP2

Timer F Compare 2 triggers Capture

HRTIM Capture Unit Trigger

HRTIM_CAPTURETRIGGER_NONE

Capture trigger is disabled

HRTIM_CAPTURETRIGGER_UPDATE

The update event triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_1

The External event 1 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_2

The External event 2 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_3

The External event 3 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_4

The External event 4 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_5

The External event 5 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_6

The External event 6 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_7

The External event 7 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_8

The External event 8 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_9

The External event 9 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_10

The External event 10 triggers the Capture

HRTIM_CAPTURETRIGGER_TA1_SET

Capture is triggered by TA1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TA1_RESET

Capture is triggered by TA1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERA_CMP1

Timer A Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERA_CMP2

Timer A Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TB1_SET

Capture is triggered by TB1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TB1_RESET

Capture is triggered by TB1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERB_CMP1

Timer B Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERB_CMP2

Timer B Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TC1_SET

Capture is triggered by TC1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TC1_RESET

Capture is triggered by TC1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERC_CMP1

Timer C Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERC_CMP2

Timer C Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TD1_SET

Capture is triggered by TD1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TD1_RESET

Capture is triggered by TD1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERD_CMP1

Timer D Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERD_CMP2

Timer D Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TE1_SET

Capture is triggered by TE1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TE1_RESET

Capture is triggered by TE1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERE_CMP1

Timer E Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERE_CMP2

Timer E Compare 2 triggers Capture

HRTIM Chopper Duty Cycle

HRTIM_CHOPPER_DUTYCYCLE_0

Only 1st pulse is present

HRTIM_CHOPPER_DUTYCYCLE_125

Duty cycle of the carrier signal is 12.5U %

HRTIM_CHOPPER_DUTYCYCLE_250

Duty cycle of the carrier signal is 25U %

HRTIM_CHOPPER_DUTYCYCLE_375

Duty cycle of the carrier signal is 37.5U %

HRTIM_CHOPPER_DUTYCYCLE_500

Duty cycle of the carrier signal is 50U %

HRTIM_CHOPPER_DUTYCYCLE_625

Duty cycle of the carrier signal is 62.5U %

HRTIM_CHOPPER_DUTYCYCLE_750

Duty cycle of the carrier signal is 75U %

HRTIM_CHOPPER_DUTYCYCLE_875

Duty cycle of the carrier signal is 87.5U %

HRTIM Chopper Frequency

HRTIM_CHOPPER_PRESCALERRATIO_DIV16

fCHPFRQ = fHRTIM / 16

HRTIM_CHOPPER_PRESCALERRATIO_DIV32

fCHPFRQ = fHRTIM / 32

HRTIM_CHOPPER_PRESCALERRATIO_DIV48

fCHPFRQ = fHRTIM / 48

HRTIM_CHOPPER_PRESCALERRATIO_DIV64

fCHPFRQ = fHRTIM / 64

HRTIM_CHOPPER_PRESCALERRATIO_DIV80

fCHPFRQ = fHRTIM / 80

HRTIM_CHOPPER_PRESCALERRATIO_DIV96

fCHPFRQ = fHRTIM / 96

HRTIM_CHOPPER_PRESCALERRATIO_DIV112

fCHPFRQ = fHRTIM / 112

HRTIM_CHOPPER_PRESCALERRATIO_DIV128

fCHPFRQ = fHRTIM / 128

HRTIM_CHOPPER_PRESCALERRATIO_DIV144

fCHPFRQ = fHRTIM / 144

HRTIM_CHOPPER_PRESCALERRATIO_DIV160

fCHPFRQ = fHRTIM / 160

HRTIM_CHOPPER_PRESCALERRATIO_DIV176

fCHPFRQ = fHRTIM / 176

HRTIM_CHOPPER_PRESCALERRATIO_DIV192

fCHPFRQ = fHRTIM / 192

HRTIM_CHOPPER_PRESCALERRATIO_DIV208

fCHPFRQ = fHRTIM / 208

HRTIM_CHOPPER_PRESCALERRATIO_DIV224

fCHPFRQ = fHRTIM / 224

HRTIM_CHOPPER_PRESCALERRATIO_DIV240

fCHPFRQ = fHRTIM / 240

HRTIM_CHOPPER_PRESCALERRATIO_DIV256

fCHPFRQ = fHRTIM / 256

HRTIM Chopper Start Pulse Width

HRTIM_CHOPPER_PULSEWIDTH_16

tSTPW = tHRTIM x 16

HRTIM_CHOPPER_PULSEWIDTH_32

tSTPW = tHRTIM x 32

HRTIM_CHOPPER_PULSEWIDTH_48

tSTPW = tHRTIM x 48

HRTIM_CHOPPER_PULSEWIDTH_64

tSTPW = tHRTIM x 64

HRTIM_CHOPPER_PULSEWIDTH_80

tSTPW = tHRTIM x 80

HRTIM_CHOPPER_PULSEWIDTH_96

tSTPW = tHRTIM x 96

HRTIM_CHOPPER_PULSEWIDTH_112

tSTPW = tHRTIM x 112

HRTIM_CHOPPER_PULSEWIDTH_128

tSTPW = tHRTIM x 128

HRTIM_CHOPPER_PULSEWIDTH_144

tSTPW = tHRTIM x 144

HRTIM_CHOPPER_PULSEWIDTH_160

tSTPW = tHRTIM x 160

HRTIM_CHOPPER_PULSEWIDTH_176

tSTPW = tHRTIM x 176

HRTIM_CHOPPER_PULSEWIDTH_192

tSTPW = tHRTIM x 192

HRTIM_CHOPPER_PULSEWIDTH_208

tSTPW = tHRTIM x 208

HRTIM_CHOPPER_PULSEWIDTH_224

tSTPW = tHRTIM x 224

HRTIM_CHOPPER_PULSEWIDTH_240

tSTPW = tHRTIM x 240

HRTIM_CHOPPER_PULSEWIDTH_256

tSTPW = tHRTIM x 256

HRTIM Common Interrupt Enable

HRTIM_IT_NONE

No interrupt enabled

HRTIM_IT_FLT1

Fault 1 interrupt enable

HRTIM_IT_FLT2

Fault 2 interrupt enable

HRTIM_IT_FLT3

Fault 3 interrupt enable

HRTIM_IT_FLT4

Fault 4 interrupt enable

HRTIM_IT_FLT5

Fault 5 interrupt enable

HRTIM_IT_FLT6

Fault 6 interrupt enable

HRTIM_IT_SYSFLT

System Fault interrupt enable

HRTIM_IT_DLLRDY

DLL ready interrupt enable

HRTIM_IT_BMPER

Burst mode period interrupt enable

HRTIM Common Interrupt Flag

HRTIM_FLAG_FLT1

Fault 1 interrupt flag

HRTIM_FLAG_FLT2

Fault 2 interrupt flag

HRTIM_FLAG_FLT3

Fault 3 interrupt flag

HRTIM_FLAG_FLT4

Fault 4 interrupt flag

HRTIM_FLAG_FLT5

Fault 5 interrupt flag

HRTIM_FLAG_FLT6

Fault 6 interrupt flag

HRTIM_FLAG_SYSFLT

System Fault interrupt flag

HRTIM_FLAG_DLLRDY

DLL ready interrupt flag

HRTIM_FLAG_BMPER

Burst mode period interrupt flag

HRTIM Compare Unit

HRTIM_COMPAREUNIT_1

Compare unit 1 identifier

HRTIM_COMPAREUNIT_2

Compare unit 2 identifier

HRTIM_COMPAREUNIT_3

Compare unit 3 identifier

HRTIM_COMPAREUNIT_4

Compare unit 4 identifier

HRTIM Compare Unit Auto Delayed Mode

HRTIM_AUTODELAYEDMODE_REGULAR

standard compare mode

HRTIM_AUTODELAYEDMODE_AUTODELAYED_NOTIMEOUT

Compare event generated only if a capture has occurred

HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP1

Compare event generated if a capture has occurred or after a Compare 1 match (timeout if capture event is missing)

HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP3

Compare event generated if a capture has occurred or after a Compare 3 match (timeout if capture event is missing)

HRTIM Counter Operating Mode

HRTIM_MODE_CONTINUOUS

The timer operates in continuous (free-running) mode

HRTIM_MODE_SINGLESHOT

The timer operates in non retriggerable single-shot mode

HRTIM_MODE_SINGLESHOT_RETRIGGERABLE

The timer operates in retriggerable single-shot mode

HRTIM Current Push Pull Status

HRTIM_PUSHPULL_CURRENTSTATUS_OUTPUT1

Signal applied on output 1 and output 2 forced inactive

HRTIM_PUSHPULL_CURRENTSTATUS_OUTPUT2

Signal applied on output 2 and output 1 forced inactive

HRTIM DAC Synchronization

HRTIM_DACSYNC_NONE

No DAC synchronization event generated

HRTIM_DACSYNC_DACTRIGOUT_1

DAC synchronization event generated on DACTrigOut1 output upon timer update

HRTIM_DACSYNC_DACTRIGOUT_2

DAC synchronization event generated on DACTrigOut2 output upon timer update

HRTIM_DACSYNC_DACTRIGOUT_3

DAC update generated on DACTrigOut3 output upon timer update

HRTIM Dead-time Falling Lock

HRTIM_TIMDEADTIME_FALLINGLOCK_WRITE

Dead-time falling value and sign is writeable

HRTIM_TIMDEADTIME_FALLINGLOCK_READONLY

Dead-time falling value and sign is read-only

HRTIM Dead-time Falling Sign

HRTIM_TIMDEADTIME_FALLINGSIGN_POSITIVE

Positive dead-time on falling edge

HRTIM_TIMDEADTIME_FALLINGSIGN_NEGATIVE

Negative dead-time on falling edge

HRTIM Dead-time Falling Sign Lock

HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_WRITE

Dead-time falling sign is writeable

HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_READONLY

Dead-time falling sign is read-only

HRTIM Dead-time Prescaler Ratio

HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL8

fDTG = fHRTIM * 8U

HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL4

fDTG = fHRTIM * 4U

HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL2

fDTG = fHRTIM * 2U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV1

fDTG = fHRTIM

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV2

fDTG = fHRTIM / 2U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV4

fDTG = fHRTIM / 4U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV8

fDTG = fHRTIM / 8U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV16

fDTG = fHRTIM / 16U

HRTIM Dead-time Rising Lock

HRTIM_TIMDEADTIME_RISINGLOCK_WRITE

Dead-time rising value and sign is writeable

HRTIM_TIMDEADTIME_RISINGLOCK_READONLY

Dead-time rising value and sign is read-only

HRTIM Dead-time Rising Sign

HRTIM_TIMDEADTIME_RISINGSIGN_POSITIVE

Positive dead-time on rising edge

HRTIM_TIMDEADTIME_RISINGSIGN_NEGATIVE

Negative dead-time on rising edge

HRTIM Dead-time Rising Sign Lock

HRTIM_TIMDEADTIME_RISINGSIGNLOCK_WRITE

Dead-time rising sign is writeable

HRTIM_TIMDEADTIME_RISINGSIGNLOCK_READONLY

Dead-time rising sign is read-only

HRTIM DLL Calibration Rate

HRTIM_SINGLE_CALIBRATION

Non periodic DLL calibration

HRTIM_CALIBRATIONRATE_0

Periodic DLL calibration: T = 1048576U * tHRTIM (6.168 ms)

HRTIM_CALIBRATIONRATE_1

Periodic DLL calibration: T = 131072U * tHRTIM (0.771 ms)

HRTIM_CALIBRATIONRATE_2

Periodic DLL calibration: T = 16384U * tHRTIM (0.096 ms)

HRTIM_CALIBRATIONRATE_3

Periodic DLL calibration: T = 2048U * tHRTIM (0.012 ms)

HRTIM Exported Macros

__HAL_HRTIM_COUNTER_MODE_UP

Description:

- configures the actual direction of the counter to UP counting mode

Parameters:

- __HANDLE__: : HRTIM handle.
- __TIMER__: : Timer index This parameter can be a combination of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return value:

- none

__HAL_HRTIM_COUNTER_MODE_UPDOWN

Description:

- configures the actual direction of the counter to UP-DOWN counting mode

Parameters:

- __HANDLE__: : HRTIM handle.
- __TIMER__: : Timer index This parameter can be a combination of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return value:

- none

__HAL_HRTIM_TIMER_OUTPUT_SWAP

Description:

- swap the output of the timer HRTIM_SETA1R and HRTIM_RSTA1R are coding for the output A2, HRTIM_SETA2R and HRTIM_RSTA2R are coding for the output A1

Parameters:

- __HANDLE__: : HRTIM handle.
- __TIMER__: : Timer index This parameter can be a combination of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return value:

- none

__HAL_HRTIM_TIMER_OUTPUT_NOSWAP

Description:

- Un-swap the output of the timer HRTIM_SETA1R and HRTIM_RSTA1R are coding for the output A1, HRTIM_SETA2R and HRTIM_RSTA2R are coding for the output A2.

Parameters:

- __HANDLE__: : HRTIM handle.
- __TIMER__: : Timer index This parameter can be a combination of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
  - HRTIM_TIMERINDEX_TIMER_F for timer F

Return value:

- none

__HAL_HRTIM_RESET_HANDLE_STATE

Description:

- Reset HRTIM handle state.

Parameters:

- __HANDLE__: HRTIM handle.

Return value:

- None

__HAL_HRTIM_ENABLE

Description:

- Enables or disables the timer counter(s)

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be any combinations of the following values:
  - HRTIM_TIMERID_MASTER: Master timer identifier
  - HRTIM_TIMERID_TIMER_A: Timer A identifier
  - HRTIM_TIMERID_TIMER_B: Timer B identifier
  - HRTIM_TIMERID_TIMER_C: Timer C identifier
  - HRTIM_TIMERID_TIMER_D: Timer D identifier
  - HRTIM_TIMERID_TIMER_E: Timer E identifier
  - HRTIM_TIMERID_TIMER_F: Timer F identifier

Return value:

- None

HRTIM_TAOEN_MASK

HRTIM_TBOEN_MASK

HRTIM_TCOEN_MASK

HRTIM_TDOEN_MASK

HRTIM_TEOEN_MASK

HRTIM_TFOEN_MASK

__HAL_HRTIM_DISABLE

__HAL_HRTIM_EXTERNAL_EVENT_COUNTER_ENABLE

Description:

- Enables the External Event counter.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A: Timer A identifier
  - HRTIM_TIMERINDEX_TIMER_B: Timer B identifier
  - HRTIM_TIMERINDEX_TIMER_C: Timer C identifier
  - HRTIM_TIMERINDEX_TIMER_D: Timer D identifier
  - HRTIM_TIMERINDEX_TIMER_E: Timer E identifier
  - HRTIM_TIMERINDEX_TIMER_F: Timer F identifier
- Event: external event Counter A or B for which timer event must be enabled This parameter can be one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B

Return value:

- None

__HAL_HRTIM_EXTERNAL_EVENT_COUNTER_DISABLE

Description:

- Disables the External Event counter.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A: Timer A identifier
  - HRTIM_TIMERINDEX_TIMER_B: Timer B identifier
  - HRTIM_TIMERINDEX_TIMER_C: Timer C identifier
  - HRTIM_TIMERINDEX_TIMER_D: Timer D identifier
  - HRTIM_TIMERINDEX_TIMER_E: Timer E identifier
  - HRTIM_TIMERINDEX_TIMER_F: Timer F identifier
- Event: external event A or B for which timer event must be disabled This parameter can be one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B

Return value:

- None

__HAL_HRTIM_EXTERNAL_EVENT_COUNTER_RESET

Description:

- Resets the External Event counter.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A: Timer A identifier
  - HRTIM_TIMERINDEX_TIMER_B: Timer B identifier
  - HRTIM_TIMERINDEX_TIMER_C: Timer C identifier
  - HRTIM_TIMERINDEX_TIMER_D: Timer D identifier
  - HRTIM_TIMERINDEX_TIMER_E: Timer E identifier
  - HRTIM_TIMERINDEX_TIMER_F: Timer F identifier
- Event: external event A or B for which timer event must be reset This parameter can be one of the following values:
  - HRTIM_TIMEEVENT_A
  - HRTIM_TIMEEVENT_B

Return value:

- None

__HAL_HRTIM_ENABLE_IT

Description:

- Enables or disables the specified HRTIM common interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 interrupt enable
  - HRTIM_IT_FLT4: Fault 4 interrupt enable
  - HRTIM_IT_FLT5: Fault 5 interrupt enable
  - HRTIM_IT_FLT6: Fault 6 interrupt enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_DLLRDY: DLL ready interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- None

__HAL_HRTIM_DISABLE_IT

__HAL_HRTIM_MASTER_ENABLE_IT

Description:

- Enables or disables the specified HRTIM Master timer interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

- None

__HAL_HRTIM_MASTER_DISABLE_IT

__HAL_HRTIM_TIMER_ENABLE_IT

Description:

- Enables or disables the specified HRTIM Timerx interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to F)
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
  - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
  - HRTIM_TIM_IT_REP: Timer repetition interrupt enable
  - HRTIM_TIM_IT_UPD: Timer update interrupt enable
  - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
  - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
  - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
  - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
  - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
  - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
  - HRTIM_TIM_IT_RST: Timer reset interrupt enable
  - HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

- None

__HAL_HRTIM_TIMER_DISABLE_IT

__HAL_HRTIM_GET_ITSTATUS

Description:

- Checks if the specified HRTIM common interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 enable
  - HRTIM_IT_FLT4: Fault 4 enable
  - HRTIM_IT_FLT5: Fault 5 enable
  - HRTIM_IT_FLT6: Fault 6 enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_DLLRDY: DLL ready interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_HRTIM_MASTER_GET_ITSTATUS

Description:

- Checks if the specified HRTIM Master interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_HRTIM_TIMER_GET_ITSTATUS

Description:

- Checks if the specified HRTIM Timerx interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to F)
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
  - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
  - HRTIM_TIM_IT_REP: Timer repetition interrupt enable
  - HRTIM_TIM_IT_UPD: Timer update interrupt enable
  - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
  - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
  - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
  - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
  - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
  - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
  - HRTIM_TIM_IT_RST: Timer reset interrupt enable
  - HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_HRTIM_CLEAR_IT

Description:

- Clears the specified HRTIM common pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt clear flag
  - HRTIM_IT_FLT2: Fault 2 interrupt clear flag
  - HRTIM_IT_FLT3: Fault 3 clear flag
  - HRTIM_IT_FLT4: Fault 4 clear flag
  - HRTIM_IT_FLT5: Fault 5 clear flag
  - HRTIM_IT_FLT6: Fault 6 clear flag
  - HRTIM_IT_SYSFLT: System Fault interrupt clear flag
  - HRTIM_IT_DLLRDY: DLL ready interrupt clear flag
  - HRTIM_IT_BMPER: Burst mode period interrupt clear flag

Return value:

- None

__HAL_HRTIM_MASTER_CLEAR_IT

Description:

- Clears the specified HRTIM Master pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt clear flag
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt clear flag
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt clear flag
  - HRTIM_MASTER_IT_MUPD: Master update interrupt clear flag

Return value:

- None

__HAL_HRTIM_TIMER_CLEAR_IT

Description:

- Clears the specified HRTIM Timerx pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to F)
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt clear flag
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt clear flag
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt clear flag
  - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt clear flag
  - HRTIM_TIM_IT_REP: Timer repetition interrupt clear flag
  - HRTIM_TIM_IT_UPD: Timer update interrupt clear flag
  - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt clear flag
  - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt clear flag
  - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt clear flag
  - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt clear flag
  - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt clear flag
  - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt clear flag
  - HRTIM_TIM_IT_RST: Timer reset interrupt clear flag
  - HRTIM_TIM_IT_DLYPRT: Timer output 1 delay protection interrupt clear flag

Return value:

- None

__HAL_HRTIM_MASTER_ENABLE_DMA

Description:

- Enables or disables the specified HRTIM Master timer DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
  - HRTIM_MASTER_DMA_MCMP1: Master compare 1 DMA request enable
  - HRTIM_MASTER_DMA_MCMP2: Master compare 2 DMA request enable
  - HRTIM_MASTER_DMA_MCMP3: Master compare 3 DMA request enable
  - HRTIM_MASTER_DMA_MCMP4: Master compare 4 DMA request enable
  - HRTIM_MASTER_DMA_MREP: Master Repetition DMA request enable
  - HRTIM_MASTER_DMA_SYNC: Synchronization input DMA request enable
  - HRTIM_MASTER_DMA_MUPD: Master update DMA request enable

Return value:

- None

__HAL_HRTIM_MASTER_DISABLE_DMA

__HAL_HRTIM_TIMER_ENABLE_DMA

Description:

- Enables or disables the specified HRTIM Timerx DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to F)
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
  - HRTIM_TIM_DMA_CMP1: Timer compare 1 DMA request enable
  - HRTIM_TIM_DMA_CMP2: Timer compare 2 DMA request enable
  - HRTIM_TIM_DMA_CMP3: Timer compare 3 DMA request enable
  - HRTIM_TIM_DMA_CMP4: Timer compare 4 DMA request enable
  - HRTIM_TIM_DMA_REP: Timer repetition DMA request enable
  - HRTIM_TIM_DMA_UPD: Timer update DMA request enable
  - HRTIM_TIM_DMA_CPT1: Timer capture 1 DMA request enable
  - HRTIM_TIM_DMA_CPT2: Timer capture 2 DMA request enable
  - HRTIM_TIM_DMA_SET1: Timer output 1 set DMA request enable
  - HRTIM_TIM_DMA_RST1: Timer output 1 reset DMA request enable
  - HRTIM_TIM_DMA_SET2: Timer output 2 set DMA request enable
  - HRTIM_TIM_DMA_RST2: Timer output 2 reset DMA request enable
  - HRTIM_TIM_DMA_RST: Timer reset DMA request enable
  - HRTIM_TIM_DMA_DLYPRT: Timer delay protection DMA request enable

Return value:

- None

__HAL_HRTIM_TIMER_DISABLE_DMA

__HAL_HRTIM_GET_FLAG

__HAL_HRTIM_CLEAR_FLAG

__HAL_HRTIM_MASTER_GET_FLAG

__HAL_HRTIM_MASTER_CLEAR_FLAG

__HAL_HRTIM_TIMER_GET_FLAG

__HAL_HRTIM_TIMER_CLEAR_FLAG

__HAL_HRTIM_SETCOUNTER

Description:

- Sets the HRTIM timer Counter Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x6 for master timer
  - 0x0 to 0x5 for timers A to F
- __COUNTER__: specifies the Counter Register new value.

Return value:

- None

__HAL_HRTIM_GETCOUNTER

Description:

- Gets the HRTIM timer Counter Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x6 for master timer
  - 0x0 to 0x5 for timers A to F

Return value:

- HRTIM: timer Counter Register value

__HAL_HRTIM_SETPERIOD

Description:

- Sets the HRTIM timer Period value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x6 for master timer
  - 0x0 to 0x5 for timers A to F
- __PERIOD__: specifies the Period Register new value.

Return value:

- None

__HAL_HRTIM_GETPERIOD

Description:

- Gets the HRTIM timer Period Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x6 for master timer
  - 0x0 to 0x5 for timers A to F

Return value:

- timer: Period Register

__HAL_HRTIM_SETCLOCKPRESCALER

Description:

- Sets the HRTIM timer clock prescaler value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x6 for master timer
  - 0x0 to 0x5 for timers A to F
- __PRESCALER__: specifies the clock prescaler new value. This parameter can be one of the following values:
  - HRTIM_PRESCALERRATIO_MUL32: fHRCK: 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL16: fHRCK: 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL8: fHRCK: 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL4: fHRCK: 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL2: fHRCK: 288 MHz - Resolution: 3.47 ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_DIV1: fHRCK: 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_DIV2: fHRCK: 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_DIV4: fHRCK: 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

Return value:

- None

__HAL_HRTIM_GETCLOCKPRESCALER

Description:

- Gets the HRTIM timer clock prescaler value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x6 for master timer
  - 0x0 to 0x5 for timers A to F

Return value:

- timer: clock prescaler value

__HAL_HRTIM_SETCOMPARE

Description:

- Sets the HRTIM timer Compare Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x0 to 0x5 for timers A to F
- __COMPAREUNIT__: timer compare unit This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4
- __COMPARE__: specifies the Compare new value.

Return value:

- None

__HAL_HRTIM_GETCOMPARE

Description:

- Gets the HRTIM timer Compare Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x0 to 0x5 for timers A to F
- __COMPAREUNIT__: timer compare unit This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4

Return value:

- Compare: value

__HAL_HRTIM_FAULT_BLANKING_ENABLE

Description:

- Enables the Fault Counter.

Parameters:

- hhrtim: pointer to HAL HRTIM handle
- Fault: fault input to enable This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6

Return value:

- HAL: status

Notes:

- This function must be called when fault is not enabled

__HAL_HRTIM_FAULT_BLANKING_DISABLE

Description:

- Disables the Fault Counter.

Parameters:

- hhrtim: pointer to HAL HRTIM handle
- Fault: fault input to disable This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
  - HRTIM_FAULT_6: Fault input 6

Return value:

- HAL: status

HRTIM External Event Channels

HRTIM_EVENT_NONE

Undefined event channel

HRTIM_EVENT_1

External event channel 1 identifier

HRTIM_EVENT_2

External event channel 2 identifier

HRTIM_EVENT_3

External event channel 3 identifier

HRTIM_EVENT_4

External event channel 4 identifier

HRTIM_EVENT_5

External event channel 5 identifier

HRTIM_EVENT_6

External event channel 6 identifier

HRTIM_EVENT_7

External event channel 7 identifier

HRTIM_EVENT_8

External event channel 8 identifier

HRTIM_EVENT_9

External event channel 9 identifier

HRTIM_EVENT_10

External event channel 10 identifier

HRTIM External Event Fast Mode

HRTIM_EVENTFASTMODE_DISABLE

External Event is re-synchronized by the HRTIM logic before acting on outputs

HRTIM_EVENTFASTMODE_ENABLE

External Event is acting asynchronously on outputs (low latency mode)

HRTIM External Event Filter

HRTIM_EVENTFILTER_NONE

Filter disabled

HRTIM_EVENTFILTER_1

fSAMPLING= fHRTIM, N=2U

HRTIM_EVENTFILTER_2

fSAMPLING= fHRTIM, N=4U

HRTIM_EVENTFILTER_3

fSAMPLING= fHRTIM, N=8U

HRTIM_EVENTFILTER_4

fSAMPLING= fEEVS/2U, N=6U

HRTIM_EVENTFILTER_5

fSAMPLING= fEEVS/2U, N=8U

HRTIM_EVENTFILTER_6

fSAMPLING= fEEVS/4U, N=6U

HRTIM_EVENTFILTER_7

fSAMPLING= fEEVS/4U, N=8U

HRTIM_EVENTFILTER_8

fSAMPLING= fEEVS/8U, N=6U

HRTIM_EVENTFILTER_9

fSAMPLING= fEEVS/8U, N=8U

HRTIM_EVENTFILTER_10

fSAMPLING= fEEVS/16U, N=5U

HRTIM_EVENTFILTER_11

fSAMPLING= fEEVS/16U, N=6U

HRTIM_EVENTFILTER_12

fSAMPLING= fEEVS/16U, N=8U

HRTIM_EVENTFILTER_13

fSAMPLING= fEEVS/32U, N=5U

HRTIM_EVENTFILTER_14

fSAMPLING= fEEVS/32U, N=6U

HRTIM_EVENTFILTER_15

fSAMPLING= fEEVS/32U, N=8U

HRTIM External Event Polarity

HRTIM_EVENTPOLARITY_HIGH

External event is active high

HRTIM_EVENTPOLARITY_LOW

External event is active low

HRTIM External Event Prescaler

HRTIM_EVENTPRESCALER_DIV1

fEEVS=fHRTIM

HRTIM_EVENTPRESCALER_DIV2

fEEVS=fHRTIM / 2U

HRTIM_EVENTPRESCALER_DIV4

fEEVS=fHRTIM / 4U

HRTIM_EVENTPRESCALER_DIV8

fEEVS=fHRTIM / 8U

HRTIM External Event Sensitivity

HRTIM_EVENTSENSITIVITY_LEVEL

External event is active on level

HRTIM_EVENTSENSITIVITY_RISINGEDGE

External event is active on Rising edge

HRTIM_EVENTSENSITIVITY_FALLINGEDGE

External event is active on Falling edge

HRTIM_EVENTSENSITIVITY_BOTHEDGES

External event is active on Rising and Falling edges

HRTIM External Event Sources

HRTIM_EEV1SRC_GPIO

External event source 1U for External Event 1

HRTIM_EEV2SRC_GPIO

External event source 1U for External Event 2

HRTIM_EEV3SRC_GPIO

External event source 1U for External Event 3

HRTIM_EEV4SRC_GPIO

External event source 1U for External Event 4

HRTIM_EEV5SRC_GPIO

External event source 1U for External Event 5

HRTIM_EEV6SRC_GPIO

External event source 1U for External Event 6

HRTIM_EEV7SRC_GPIO

External event source 1U for External Event 7

HRTIM_EEV8SRC_GPIO

External event source 1U for External Event 8

HRTIM_EEV9SRC_GPIO

External event source 1U for External Event 9

HRTIM_EEV10SRC_GPIO

External event source 1U for External Event 10

HRTIM_EEV1SRC_COMP2_OUT

External event source 2U for External Event 1

HRTIM_EEV2SRC_COMP4_OUT

External event source 2U for External Event 2

HRTIM_EEV3SRC_COMP6_OUT

External event source 2U for External Event 3

HRTIM_EEV4SRC_COMP1_OUT

External event source 2U for External Event 4

HRTIM_EEV5SRC_COMP3_OUT

External event source 2U for External Event 5

HRTIM_EEV6SRC_COMP2_OUT

External event source 2U for External Event 6

HRTIM_EEV7SRC_COMP4_OUT

External event source 2U for External Event 7

HRTIM_EEV8SRC_COMP6_OUT

External event source 2U for External Event 8

HRTIM_EEV9SRC_COMP5_OUT

External event source 2U for External Event 9

HRTIM_EEV10SRC_COMP7_OUT

External event source 2U for External Event 10

HRTIM_EEV1SRC_TIM1_TRGO

External event source 3U for External Event 1

HRTIM_EEV2SRC_TIM2_TRGO

External event source 3U for External Event 2

HRTIM_EEV3SRC_TIM3_TRGO

External event source 3U for External Event 3

HRTIM_EEV4SRC_COMP5_OUT

External event source 3U for External Event 4

HRTIM_EEV5SRC_COMP7_OUT

External event source 3U for External Event 5

HRTIM_EEV6SRC_COMP1_OUT

External event source 3U for External Event 6

HRTIM_EEV7SRC_TIM7_TRGO

External event source 3U for External Event 7

HRTIM_EEV8SRC_COMP3_OUT

External event source 3U for External Event 8

HRTIM_EEV9SRC_TIM15_TRGO

External event source 3U for External Event 9

HRTIM_EEV10SRC_TIM6_TRGO

External event source 3U for External Event 10

HRTIM_EEV1SRC_ADC1_AWD1

External event source 4U for External Event 1

HRTIM_EEV2SRC_ADC1_AWD2

External event source 4U for External Event 2

HRTIM_EEV3SRC_ADC1_AWD3

External event source 4U for External Event 3

HRTIM_EEV4SRC_ADC2_AWD1

External event source 4U for External Event 4

HRTIM_EEV5SRC_ADC2_AWD2

External event source 4U for External Event 5

HRTIM_EEV6SRC_ADC2_AWD3

External event source 4U for External Event 6

HRTIM_EEV7SRC_ADC3_AWD1

External event source 4U for External Event 7

HRTIM_EEV8SRC_ADC4_AWD1

External event source 4U for External Event 8

HRTIM_EEV9SRC_COMP4_OUT

External event source 4U for External Event 9

HRTIM_EEV10SRC_ADC5_AWD1

External event source 4U for External Event 10

HRTIM External Fault Prescaler

HRTIM_FAULTPRESCALER_DIV1

fFLTS=fHRTIM

HRTIM_FAULTPRESCALER_DIV2

fFLTS=fHRTIM / 2U

HRTIM_FAULTPRESCALER_DIV4

fFLTS=fHRTIM / 4U

HRTIM_FAULTPRESCALER_DIV8

fFLTS=fHRTIM / 8U

HRTIM Fault Blanking Source

HRTIM_FAULTBLANKINGMODE_RSTALIGNED

Fault blanking source is Reset-aligned window

HRTIM_FAULTBLANKINGMODE_MOVING

Fault blanking source is Moving window

HRTIM Fault Blanking Control

HRTIM_FAULTBLANKINGCTL_DISABLED

No blanking on Fault

HRTIM_FAULTBLANKINGCTL_ENABLED

Fault blanking mode

HRTIM Fault Channel

HRTIM_FAULT_1

Fault channel 1 identifier

HRTIM_FAULT_2

Fault channel 2 identifier

HRTIM_FAULT_3

Fault channel 3 identifier

HRTIM_FAULT_4

Fault channel 4 identifier

HRTIM_FAULT_5

Fault channel 5 identifier

HRTIM_FAULT_6

Fault channel 6 identifier

HRTIM Fault counter threshold value

HRTIM_FAULTCOUNTER_NONE

Counter threshold = 0U

HRTIM_FAULTCOUNTER_1

Counter threshold = 1U

HRTIM_FAULTCOUNTER_2

Counter threshold = 2U

HRTIM_FAULTCOUNTER_3

Counter threshold = 3U

HRTIM_FAULTCOUNTER_4

Counter threshold = 4U

HRTIM_FAULTCOUNTER_5

Counter threshold = 5U

HRTIM_FAULTCOUNTER_6

Counter threshold = 6U

HRTIM_FAULTCOUNTER_7

Counter threshold = 7U

HRTIM_FAULTCOUNTER_8

Counter threshold = 8U

HRTIM_FAULTCOUNTER_9

Counter threshold = 9U

HRTIM_FAULTCOUNTER_10

Counter threshold = 10U

HRTIM_FAULTCOUNTER_11

Counter threshold = 11U

HRTIM_FAULTCOUNTER_12

Counter threshold = 12U

HRTIM_FAULTCOUNTER_13

Counter threshold = 13U

HRTIM_FAULTCOUNTER_14

Counter threshold = 14U

HRTIM_FAULTCOUNTER_15

Counter threshold = 15U

HRTIM Fault Filter

HRTIM_FAULTFILTER_NONE

Filter disabled

HRTIM_FAULTFILTER_1

fSAMPLING= fHRTIM, N=2U

HRTIM_FAULTFILTER_2

fSAMPLING= fHRTIM, N=4U

HRTIM_FAULTFILTER_3

fSAMPLING= fHRTIM, N=8U

HRTIM_FAULTFILTER_4

fSAMPLING= fFLTS/2U, N=6U

HRTIM_FAULTFILTER_5

fSAMPLING= fFLTS/2U, N=8U

HRTIM_FAULTFILTER_6

fSAMPLING= fFLTS/4U, N=6U

HRTIM_FAULTFILTER_7

fSAMPLING= fFLTS/4U, N=8U

HRTIM_FAULTFILTER_8

fSAMPLING= fFLTS/8U, N=6U

HRTIM_FAULTFILTER_9

fSAMPLING= fFLTS/8U, N=8U

HRTIM_FAULTFILTER_10

fSAMPLING= fFLTS/16U, N=5U

HRTIM_FAULTFILTER_11

fSAMPLING= fFLTS/16U, N=6U

HRTIM_FAULTFILTER_12

fSAMPLING= fFLTS/16U, N=8U

HRTIM_FAULTFILTER_13

fSAMPLING= fFLTS/32U, N=5U

HRTIM_FAULTFILTER_14

fSAMPLING= fFLTS/32U, N=6U

HRTIM_FAULTFILTER_15

fSAMPLING= fFLTS/32U, N=8U

HRTIM Fault Input Sources

HRTIM_FLTINR1_FLT1SRC

bit 0 of the source input for Fault channel 1

HRTIM_FLTINR1_FLT2SRC

bit 0 of the source input for Fault channel 2

HRTIM_FLTINR1_FLT3SRC

bit 0 of the source input for Fault channel 3

HRTIM_FLTINR1_FLT4SRC

bit 0 of the source input for Fault channel 4

HRTIM_FLTINR2_FLT5SRC

bit 0 of the source input for Fault channel 5

HRTIM_FLTINR2_FLT6SRC

bit 0 of the source input for Fault channel 6

HRTIM Fault Lock

HRTIM_FAULTLOCK_READWRITE

Fault settings bits are read/write

HRTIM_FAULTLOCK_READONLY

Fault settings bits are read only

HRTIM Fault Mode Control

HRTIM_FAULTMODECTL_DISABLED

Fault channel is disabled

HRTIM_FAULTMODECTL_ENABLED

Fault channel is enabled

HRTIM Fault Polarity

HRTIM_FAULTPOLARITY_LOW

Fault input is active low

HRTIM_FAULTPOLARITY_HIGH

Fault input is active high

HRTIM Fault Reset Mode

HRTIM_FAULTCOUNTERRST_UNCONDITIONAL

Fault counter is reset on each reset / roll-over event

HRTIM_FAULTCOUNTERRST_CONDITIONAL

Fault counter is reset on each reset / roll-over event only if no fault occurred during last countingperiod.

HRTIM Fault Sources

HRTIM_FAULTSOURCE_DIGITALINPUT

Fault input is FLT input pin

HRTIM_FAULTSOURCE_INTERNAL

Fault input is FLT_Int signal (e.g. internal comparator)

HRTIM_FAULTSOURCE_EEVINPUT

Fault input is EEV pin

HRTIM Half Mode Enable

HRTIM_HALFMODE_DISABLED

Half mode is disabled

HRTIM_HALFMODE_ENABLED

Half mode is enabled

HRTIM Idle Push Pull Status

HRTIM_PUSHPULL_IDLESTATUS_OUTPUT1

Protection occurred when the output 1 was active and output 2 forced inactive

HRTIM_PUSHPULL_IDLESTATUS_OUTPUT2

Protection occurred when the output 2 was active and output 1 forced inactive

HRTIM Interleaved Mode

HRTIM_INTERLEAVED_MODE_DISABLED

HRTIM interleaved Mode is disabled

HRTIM_INTERLEAVED_MODE_DUAL

HRTIM interleaved Mode is Half

HRTIM_INTERLEAVED_MODE_TRIPLE

HRTIM interleaved Mode is Triple

HRTIM_INTERLEAVED_MODE_QUAD

HRTIM interleaved Mode is Quad

HRTIM Master DMA Request Enable

HRTIM_MASTER_DMA_NONE

No DMA request enable

HRTIM_MASTER_DMA_MCMP1

Master compare 1 DMA request enable

HRTIM_MASTER_DMA_MCMP2

Master compare 2 DMA request enable

HRTIM_MASTER_DMA_MCMP3

Master compare 3 DMA request enable

HRTIM_MASTER_DMA_MCMP4

Master compare 4 DMA request enable

HRTIM_MASTER_DMA_MREP

Master Repetition DMA request enable

HRTIM_MASTER_DMA_SYNC

Synchronization input DMA request enable

HRTIM_MASTER_DMA_MUPD

Master update DMA request enable

HRTIM Master Interrupt Enable

HRTIM_MASTER_IT_NONE

No interrupt enabled

HRTIM_MASTER_IT_MCMP1

Master compare 1 interrupt enable

HRTIM_MASTER_IT_MCMP2

Master compare 2 interrupt enable

HRTIM_MASTER_IT_MCMP3

Master compare 3 interrupt enable

HRTIM_MASTER_IT_MCMP4

Master compare 4 interrupt enable

HRTIM_MASTER_IT_MREP

Master Repetition interrupt enable

HRTIM_MASTER_IT_SYNC

Synchronization input interrupt enable

HRTIM_MASTER_IT_MUPD

Master update interrupt enable

HRTIM Master Interrupt Flag

HRTIM_MASTER_FLAG_MCMP1

Master compare 1 interrupt flag

HRTIM_MASTER_FLAG_MCMP2

Master compare 2 interrupt flag

HRTIM_MASTER_FLAG_MCMP3

Master compare 3 interrupt flag

HRTIM_MASTER_FLAG_MCMP4

Master compare 4 interrupt flag

HRTIM_MASTER_FLAG_MREP

Master Repetition interrupt flag

HRTIM_MASTER_FLAG_SYNC

Synchronization input interrupt flag

HRTIM_MASTER_FLAG_MUPD

Master update interrupt flag

HRTIM Max Timer

MAX_HRTIM_TIMER

HRTIM Output Balanced Idle Automatic Resume

HRTIM_OUTPUTBIAR_DISABLED

output is not automatically re-enabled

HRTIM_OUTPUTBIAR_ENABLED

output is automatically re-enabled

HRTIM Output Burst Mode Entry Delayed

HRTIM_OUTPUTBURSTMODEENTRY_REGULAR

The programmed Idle state is applied immediately to the Output

HRTIM_OUTPUTBURSTMODEENTRY_DELAYED

Dead-time is inserted on output before entering the idle mode

HRTIM Output Chopper Mode Enable

HRTIM_OUTPUTCHOPPERMODE_DISABLED

Output signal is not altered

HRTIM_OUTPUTCHOPPERMODE_ENABLED

Output signal is chopped by a carrier signal

HRTIM Output FAULT Level

HRTIM_OUTPUTFAULTLEVEL_NONE

The output is not affected by the fault input

HRTIM_OUTPUTFAULTLEVEL_ACTIVE

Output at active level when in FAULT state

HRTIM_OUTPUTFAULTLEVEL_INACTIVE

Output at inactive level when in FAULT state

HRTIM_OUTPUTFAULTLEVEL_HIGHZ

Output is tri-stated when in FAULT state

HRTIM Output IDLE Level

HRTIM_OUTPUTIDLELEVEL_INACTIVE

Output at inactive level when in IDLE state

HRTIM_OUTPUTIDLELEVEL_ACTIVE

Output at active level when in IDLE state

HRTIM Output Idle Mode

HRTIM_OUTPUTIDLEMODE_NONE

The output is not affected by the burst mode operation

HRTIM_OUTPUTIDLEMODE_IDLE

The output is in idle state when requested by the burst mode controller

HRTIM Output Level

HRTIM_OUTPUTLEVEL_ACTIVE

Force the output to its active state

HRTIM_OUTPUTLEVEL_INACTIVE

Force the output to its inactive state

IS_HRTIM_OUTPUTLEVEL

HRTIM Output Polarity

HRTIM_OUTPUTPOLARITY_HIGH

Output is acitve HIGH

HRTIM_OUTPUTPOLARITY_LOW

Output is active LOW

HRTIM Output Reset Source

HRTIM_OUTPUTRESET_NONE

Reset the output reset crossbar

HRTIM_OUTPUTRESET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMPER

Timer period event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP1

Timer compare 1 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP2

Timer compare 2 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP3

Timer compare 3 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP4

Timer compare 4 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERPER

The master timer period event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP1

Master Timer compare 1 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP2

Master Timer compare 2 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP3

Master Timer compare 3 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP4

Master Timer compare 4 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV1_TIMBCMP1

Timer event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV2_TIMBCMP2

Timer event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV3_TIMCCMP2

Timer event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV4_TIMCCMP3

Timer event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV5_TIMDCMP1

Timer event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV6_TIMDCMP2

Timer event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV7_TIMECMP3

Timer event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV8_TIMECMP4

Timer event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMAEV9_TIMFCMP4

Timer event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV1_TIMACMP1

Timer event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV2_TIMACMP2

Timer event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV3_TIMCCMP3

Timer event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV4_TIMCCMP4

Timer event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV5_TIMDCMP3

Timer event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV6_TIMDCMP4

Timer event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV7_TIMECMP1

Timer event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV8_TIMECMP2

Timer event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMBEV9_TIMFCMP3

Timer event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV1_TIMACMP2

Timer event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV2_TIMACMP3

Timer event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV3_TIMBCMP2

Timer event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV4_TIMBCMP3

Timer event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV5_TIMDCMP2

Timer event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV6_TIMDCMP4

Timer event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV7_TIMECMP3

Timer event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV8_TIMECMP4

Timer event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCEV9_TIMFCMP2

Timer event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV1_TIMACMP1

Timer event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV2_TIMACMP4

Timer event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV3_TIMBCMP2

Timer event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV4_TIMBCMP4

Timer event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV5_TIMCCMP4

Timer event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV6_TIMECMP1

Timer event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV7_TIMECMP4

Timer event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV8_TIMFCMP1

Timer event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMDEV9_TIMFCMP3

Timer event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV1_TIMACMP4

Timer event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV2_TIMBCMP3

Timer event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV3_TIMBCMP4

Timer event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV4_TIMCCMP1

Timer event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV5_TIMCCMP2

Timer event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV6_TIMDCMP1

Timer event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV7_TIMDCMP2

Timer event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV8_TIMFCMP3

Timer event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEEV9_TIMFCMP4

Timer event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV1_TIMACMP3

Timer event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV2_TIMBCMP1

Timer event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV3_TIMBCMP4

Timer event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV4_TIMCCMP1

Timer event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV5_TIMCCMP4

Timer event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV6_TIMDCMP3

Timer event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV7_TIMDCMP4

Timer event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV8_TIMECMP2

Timer event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMFEV9_TIMECMP3

Timer event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_1

External event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_2

External event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_3

External event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_4

External event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_5

External event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_6

External event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_7

External event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_8

External event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_9

External event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_10

External event 10 forces the output to its inactive state

HRTIM_OUTPUTRESET_UPDATE

Timer register update event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_NONE

Reset the output reset crossbar

LL_HRTIM_OUTPUTRESET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMPER

Timer period event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCMP1

Timer compare 1 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCMP2

Timer compare 2 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCMP3

Timer compare 3 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCMP4

Timer compare 4 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_MASTERPER

The master timer period event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_MASTERCMP1

Master Timer compare 1 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_MASTERCMP2

Master Timer compare 2 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_MASTERCMP3

Master Timer compare 3 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_MASTERCMP4

Master Timer compare 4 event forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV1_TIMBCMP1

Timer event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV2_TIMBCMP2

Timer event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV3_TIMFCMP4

Timer event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV4_TIMCCMP2

Timer event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV5_TIMCCMP3

Timer event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV6_TIMDCMP1

Timer event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV7_TIMDCMP2

Timer event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV8_TIMECMP3

Timer event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMAEV9_TIMECMP4

Timer event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV1_TIMACMP1

Timer event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV2_TIMACMP2

Timer event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV3_TIMFCMP3

Timer event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV4_TIMCCMP3

Timer event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV5_TIMCCMP4

Timer event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV6_TIMDCMP3

Timer event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV7_TIMDCMP4

Timer event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV8_TIMECMP1

Timer event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMBEV9_TIMECMP2

Timer event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV1_TIMACMP2

Timer event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV2_TIMACMP3

Timer event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV3_TIMBCMP2

Timer event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV4_TIMBCMP3

Timer event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV5_TIMDCMP2

Timer event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV6_TIMDCMP4

Timer event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV7_TIMFCMP2

Timer event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV8_TIMECMP3

Timer event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMCEV9_TIMECMP4

Timer event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV1_TIMACMP1

Timer event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV2_TIMACMP4

Timer event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV3_TIMBCMP2

Timer event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV4_TIMBCMP4

Timer event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV5_TIMFCMP1

Timer event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV6_TIMFCMP3

Timer event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV7_TIMCCMP4

Timer event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV8_TIMECMP1

Timer event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMDEV9_TIMECMP4

Timer event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV1_TIMFCMP3

Timer event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV2_TIMACMP4

Timer event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV3_TIMBCMP3

Timer event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV4_TIMBCMP4

Timer event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV5_TIMCCMP1

Timer event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV6_TIMCCMP2

Timer event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV7_TIMDCMP1

Timer event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV8_TIMDCMP2

Timer event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMEEV9_TIMFCMP4

Timer event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV1_TIMACMP3

Timer event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV2_TIMBCMP1

Timer event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV3_TIMBCMP4

Timer event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV4_TIMCCMP1

Timer event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV5_TIMCCMP4

Timer event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV6_TIMDCMP3

Timer event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV7_TIMDCMP4

Timer event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV8_TIMECMP2

Timer event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_TIMFEV9_TIMECMP3

Timer event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_1

External event 1 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_2

External event 2 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_3

External event 3 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_4

External event 4 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_5

External event 5 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_6

External event 6 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_7

External event 7 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_8

External event 8 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_9

External event 9 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_EEV_10

External event 10 forces the output to its inactive state

LL_HRTIM_OUTPUTRESET_UPDATE

Timer register update event forces the output to its inactive state

HRTIM Output Set Source

HRTIM_OUTPUTSET_NONE

Reset the output set crossbar

HRTIM_OUTPUTSET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its active state

HRTIM_OUTPUTSET_TIMPER

Timer period event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP1

Timer compare 1 event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP2

Timer compare 2 event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP3

Timer compare 3 event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP4

Timer compare 4 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERPER

The master timer period event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP1

Master Timer compare 1 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP2

Master Timer compare 2 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP3

Master Timer compare 3 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP4

Master Timer compare 4 event forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV1_TIMBCMP1

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV2_TIMBCMP2

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV3_TIMCCMP2

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV4_TIMCCMP3

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV5_TIMDCMP1

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV6_TIMDCMP2

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV7_TIMECMP3

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV8_TIMECMP4

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMAEV9_TIMFCMP4

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV1_TIMACMP1

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV2_TIMACMP2

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV3_TIMCCMP3

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV4_TIMCCMP4

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV5_TIMDCMP3

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV6_TIMDCMP4

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV7_TIMECMP1

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV8_TIMECMP2

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMBEV9_TIMFCMP3

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV1_TIMACMP2

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV2_TIMACMP3

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV3_TIMBCMP2

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV4_TIMBCMP3

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV5_TIMDCMP2

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV6_TIMDCMP4

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV7_TIMECMP3

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV8_TIMECMP4

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMCEV9_TIMFCMP2

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV1_TIMACMP1

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV2_TIMACMP4

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV3_TIMBCMP2

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV4_TIMBCMP4

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV5_TIMCCMP4

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV6_TIMECMP1

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV7_TIMECMP4

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV8_TIMFCMP1

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMDEV9_TIMFCMP3

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV1_TIMACMP4

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV2_TIMBCMP3

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV3_TIMBCMP4

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV4_TIMCCMP1

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV5_TIMCCMP2

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV6_TIMDCMP1

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV7_TIMDCMP2

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV8_TIMFCMP3

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMEEV9_TIMFCMP4

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV1_TIMACMP3

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV2_TIMBCMP1

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV3_TIMBCMP4

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV4_TIMCCMP1

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV5_TIMCCMP4

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV6_TIMDCMP3

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV7_TIMDCMP4

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV8_TIMECMP2

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMFEV9_TIMECMP3

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_EEV_1

External event 1 forces the output to its active state

HRTIM_OUTPUTSET_EEV_2

External event 2 forces the output to its active state

HRTIM_OUTPUTSET_EEV_3

External event 3 forces the output to its active state

HRTIM_OUTPUTSET_EEV_4

External event 4 forces the output to its active state

HRTIM_OUTPUTSET_EEV_5

External event 5 forces the output to its active state

HRTIM_OUTPUTSET_EEV_6

External event 6 forces the output to its active state

HRTIM_OUTPUTSET_EEV_7

External event 7 forces the output to its active state

HRTIM_OUTPUTSET_EEV_8

External event 8 forces the output to its active state

HRTIM_OUTPUTSET_EEV_9

External event 9 forces the output to its active state

HRTIM_OUTPUTSET_EEV_10

External event 10 forces the output to its active state

HRTIM_OUTPUTSET_UPDATE

Timer register update event forces the output to its active state

HRTIM Output State

HRTIM_OUTPUTSTATE_IDLE

Main operating mode, where the output can take the active or inactive level as programmed in the crossbar unit

HRTIM_OUTPUTSTATE_RUN

Default operating state (e.g. after an HRTIM reset, when the outputs are disabled by software or during a burst mode operation

HRTIM_OUTPUTSTATE_FAULT

Safety state, entered in case of a shut-down request on FAULTx inputs

HRTIM Prescaler Ratio

HRTIM_PRESCALERRATIO_MUL32

fHRCK: fHRTIM x 32U = 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_MUL16

fHRCK: fHRTIM x 16U = 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_MUL8

fHRCK: fHRTIM x 8U = 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_MUL4

fHRCK: fHRTIM x 4U = 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_MUL2

fHRCK: fHRTIM x 2U = 288 MHz - Resolution: 3.47 ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_DIV1

fHRCK: fHRTIM = 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_DIV2

fHRCK: fHRTIM / 2U = 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_DIV4

fHRCK: fHRTIM / 4U = 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

HRTIM Register Preload Enable

HRTIM_PRELOAD_DISABLED

Preload disabled: the write access is directly done into the active register

HRTIM_PRELOAD_ENABLED

Preload enabled: the write access is done into the preload register

HRTIM Reset On Sync Input Event

HRTIM_SYNCRESET_DISABLED

Synchronization input event has effect on the timer

HRTIM_SYNCRESET_ENABLED

Synchronization input event resets the timer

HRTIM Simple OC Mode

HRTIM_BASICOCMODE_TOGGLE

Output toggles when the timer counter reaches the compare value

HRTIM_BASICOCMODE_INACTIVE

Output forced to active level when the timer counter reaches the compare value

HRTIM_BASICOCMODE_ACTIVE

Output forced to inactive level when the timer counter reaches the compare value

IS_HRTIM_BASICOCMODE

HRTIM Software Timer Reset

HRTIM_TIMERRESET_MASTER

Reset the master timer counter

HRTIM_TIMERRESET_TIMER_A

Reset the timer A counter

HRTIM_TIMERRESET_TIMER_B

Reset the timer B counter

HRTIM_TIMERRESET_TIMER_C

Reset the timer C counter

HRTIM_TIMERRESET_TIMER_D

Reset the timer D counter

HRTIM_TIMERRESET_TIMER_E

Reset the timer E counter

HRTIM_TIMERRESET_TIMER_F

Reset the timer F counter

HRTIM Software Timer swap Output

HRTIM_TIMERSWAP_A

Swap the output of the Timer A

HRTIM_TIMERSWAP_B

Swap the output of the Timer B

HRTIM_TIMERSWAP_C

Swap the output of the Timer C

HRTIM_TIMERSWAP_D

Swap the output of the Timer D

HRTIM_TIMERSWAP_E

Swap the output of the Timer E

HRTIM_TIMERSWAP_F

Swap the output of the Timer F

HRTIM Software Timer Update

HRTIM_TIMERUPDATE_MASTER

Force an immediate transfer from the preload to the active register in the master timer

HRTIM_TIMERUPDATE_A

Force an immediate transfer from the preload to the active register in the timer A

HRTIM_TIMERUPDATE_B

Force an immediate transfer from the preload to the active register in the timer B

HRTIM_TIMERUPDATE_C

Force an immediate transfer from the preload to the active register in the timer C

HRTIM_TIMERUPDATE_D

Force an immediate transfer from the preload to the active register in the timer D

HRTIM_TIMERUPDATE_E

Force an immediate transfer from the preload to the active register in the timer E

HRTIM_TIMERUPDATE_F

Forces an immediate transfer from the preload to the active register in the timer F

HRTIM Start On Sync Input Event

HRTIM_SYNCSTART_DISABLED

Synchronization input event has effect on the timer

HRTIM_SYNCSTART_ENABLED

Synchronization input event starts the timer

HRTIM Synchronization Input Source

HRTIM_SYNCINPUTSOURCE_NONE

disabled. HRTIM is not synchronized and runs in standalone mode

HRTIM_SYNCINPUTSOURCE_INTERNALEVENT

The HRTIM is synchronized with the on-chip timer

HRTIM_SYNCINPUTSOURCE_EXTERNALEVENT

A positive pulse on SYNCIN input triggers the HRTIM

HRTIM Synchronization Options

HRTIM_SYNCOPTION_NONE

HRTIM instance doesn't handle external synchronization signals (SYNCIN, SYNCOUT)

HRTIM_SYNCOPTION_MASTER

HRTIM instance acts as a MASTER, i.e. generates external synchronization output (SYNCOUT)

HRTIM_SYNCOPTION_SLAVE

HRTIM instance acts as a SLAVE, i.e. it is synchronized by external sources (SYNCIN)

HRTIM Synchronization Output Polarity

HRTIM_SYNCOUTPUTPOLARITY_NONE

Synchronization output event is disabled

HRTIM_SYNCOUTPUTPOLARITY_POSITIVE

SCOUT pin has a low idle level and issues a positive pulse of 16 fHRTIM clock cycles length for the synchronization

HRTIM_SYNCOUTPUTPOLARITY_NEGATIVE

SCOUT pin has a high idle level and issues a negative pulse of 16 fHRTIM clock cycles length for the synchronization

HRTIM Synchronization Output Source

HRTIM_SYNCOUTPUTSOURCE_MASTER_START

A pulse is sent on the SYNCOUT output upon master timer start event

HRTIM_SYNCOUTPUTSOURCE_MASTER_CMP1

A pulse is sent on the SYNCOUT output upon master timer compare 1 event

HRTIM_SYNCOUTPUTSOURCE_TIMA_START

A pulse is sent on the SYNCOUT output upon timer A start or reset events

HRTIM_SYNCOUTPUTSOURCE_TIMA_CMP1

A pulse is sent on the SYNCOUT output upon timer A compare 1 event

HRTIM Timer Burst Mode

HRTIM_TIMERBURSTMODE_MAINTAINCLOCK

Timer counter clock is maintained and the timer operates normally

HRTIM_TIMERBURSTMODE_RESETCOUNTER

Timer counter clock is stopped and the counter is reset

HRTIM Timer Dead-time Insertion

HRTIM_TIMDEADTIMEINSERTION_DISABLED

Output 1 and output 2 signals are independent

HRTIM_TIMDEADTIMEINSERTION_ENABLED

Dead-time is inserted between output 1 and output 2U

HRTIM Timer Delayed Protection Mode

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DISABLED

No action

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT1_EEV6

Timers A, B, C: Output 1 delayed Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT2_EEV6

Timers A, B, C: Output 2 delayed Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDBOTH_EEV6

Timers A, B, C: Output 1 and output 2 delayed Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_BALANCED_EEV6

Timers A, B, C: Balanced Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT1_DEEV7

Timers A, B, C: Output 1 delayed Idle on external Event 7U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT2_DEEV7

Timers A, B, C: Output 2 delayed Idle on external Event 7U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDBOTH_EEV7

Timers A, B, C: Output 1 and output2 delayed Idle on external Event 7U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_BALANCED_EEV7

Timers A, B, C: Balanced Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DISABLED

No action

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT1_EEV8

Timers D, E: Output 1 delayed Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT2_EEV8

Timers D, E: Output 2 delayed Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDBOTH_EEV8

Timers D, E: Output 1 and output 2 delayed Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_BALANCED_EEV8

Timers D, E: Balanced Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT1_DEEV9

Timers D, E: Output 1 delayed Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT2_DEEV9

Timers D, E: Output 2 delayed Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDBOTH_EEV9

Timers D, E: Output 1 and output2 delayed Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_BALANCED_EEV9

Timers D, E: Balanced Idle on external Event 7U

HRTIM_TIMER_F_DELAYEDPROTECTION_DISABLED

No action

HRTIM_TIMER_F_DELAYEDPROTECTION_DELAYEDOUT1_EEV8

Timers F: Output 1 delayed Idle on external Event 6U

HRTIM_TIMER_F_DELAYEDPROTECTION_DELAYEDOUT2_EEV8

Timers F: Output 2 delayed Idle on external Event 6U

HRTIM_TIMER_F_DELAYEDPROTECTION_DELAYEDBOTH_EEV8

Timers F: Output 1 and output 2 delayed Idle on external Event 6U

HRTIM_TIMER_F_DELAYEDPROTECTION_BALANCED_EEV8

Timers F: Balanced Idle on external Event 6U

HRTIM_TIMER_F_DELAYEDPROTECTION_DELAYEDOUT1_DEEV9

Timers F: Output 1 delayed Idle on external Event 7U

HRTIM_TIMER_F_DELAYEDPROTECTION_DELAYEDOUT2_DEEV9

Timers F: Output 2 delayed Idle on external Event 7U

HRTIM_TIMER_F_DELAYEDPROTECTION_DELAYEDBOTH_EEV9

Timers F: Output 1 and output2 delayed Idle on external Event 7U

HRTIM_TIMER_F_DELAYEDPROTECTION_BALANCED_EEV9

Timers F: Balanced Idle on external Event 7U

HRTIM Dual Channel DAC Trigger Enable

HRTIM_TIMER_DCDE_DISABLED

the Dual channel DAC trigger is disabled

HRTIM_TIMER_DCDE_ENABLED

the Dual channel DAC trigger is enabled

HRTIM Dual Channel Dac Reset Trigger

HRTIM_TIMER_DCDR_COUNTER

the trigger is generated on counter reset or roll-over event

HRTIM_TIMER_DCDR_OUT1SET

the trigger is generated on output 1 set event

HRTIM Dual Channel Dac Step Trigger

HRTIM_TIMER_DCDS_CMP2

the trigger is generated on compare 2 event

HRTIM_TIMER_DCDS_OUT1RST

the trigger is generated on output 1 reset event

HRTIM Timer External Event Counter A or B

HRTIM_TIMEEVENT_A

External Event Counter A

HRTIM_TIMEEVENT_B

External Event Counter B

HRTIM Timer External Event Counter

HRTIM_TIMEEVENTCOUNTER_DISABLED

External Event Counter disabled

HRTIM_TIMEEVENTCOUNTER_ENABLED

External Event Counter enabled

HRTIM Timer External Event Filter

HRTIM_TIMEEVFLT_NONE

HRTIM_TIMEEVFLT_BLANKINGCMP1

Blanking from counter reset/roll-over to Compare 1U

HRTIM_TIMEEVFLT_BLANKINGCMP2

Blanking from counter reset/roll-over to Compare 2U

HRTIM_TIMEEVFLT_BLANKINGCMP3

Blanking from counter reset/roll-over to Compare 3U

HRTIM_TIMEEVFLT_BLANKINGCMP4

Blanking from counter reset/roll-over to Compare 4U

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF1_TIMBCMP1

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF2_TIMBCMP4

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF3_TIMBOUT2

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF4_TIMCCMP1

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF5_TIMCCMP4

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF6_TIMFCMP1

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF7_TIMDCMP1

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEEVFLT_BLANKING_TIMAEEF8_TIMECMP2

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF1_TIMACMP1

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF2_TIMACMP4

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF3_TIMAOUT2

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF4_TIMCCMP1

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF5_TIMCCMP2

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF6_TIMFCMP2

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF7_TIMDCMP2

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEEVFLT_BLANKING_TIMBEEF8_TIMECMP1

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF1_TIMACMP2

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF2_TIMBCMP1

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF3_TIMBCMP4

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF4_TIMFCMP1

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF5_TIMDCMP1

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF6_TIMDCMP4

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF7_TIMDOUT2

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEEVFLT_BLANKING_TIMCEEF8_TIMECMP4

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF1_TIMACMP1

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF2_TIMBCMP2

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF3_TIMCCMP1

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF4_TIMCCMP2

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF5_TIMCOUT2

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF6_TIMECMP1

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF7_TIMECMP4

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEEVFLT_BLANKING_TIMDEEF8_TIMFCMP4

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF1_TIMACMP2

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF2_TIMBCMP1

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF3_TIMCCMP1

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF4_TIMFCMP4

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF5_TIMFOUT2

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF6_TIMDCMP1

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF7_TIMDCMP4

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEEVFLT_BLANKING_TIMEEEF8_TIMDOUT2

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF1_TIMACMP4

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF2_TIMBCMP2

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF3_TIMCCMP4

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF4_TIMDCMP2

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF5_TIMDCMP4

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF6_TIMECMP1

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF7_TIMECMP4

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEEVFLT_BLANKING_TIMFEEF8_TIMEOUT2

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEEVFLT_WINDOWINGCMP2

Windowing from counter reset/roll-over to Compare 2U

HRTIM_TIMEEVFLT_WINDOWINGCMP3

Windowing from counter reset/roll-over to Compare 3U

HRTIM_TIMEEVFLT_WINDOWINGTIM

Windowing from another timing unit: TIMWIN source

HRTIM Timer External Event Latch

HRTIM_TIMEVENTLATCH_DISABLED

Event is ignored if it happens during a blank, or passed through during a window

HRTIM_TIMEVENTLATCH_ENABLED

Event is latched and delayed till the end of the blanking or windowing period

HRTIM Timer External Counter Reset Mode

HRTIM_TIMEEVENTRESETMODE_UNCONDITIONAL

External Event Counter is reset on each reset / roll-over event

HRTIM_TIMEEVENTRESETMODE_CONDITIONAL

External Event Counter is reset on each reset / roll-over event only if no event occurs during last counting period

HRTIM Timer Fault Enabling

HRTIM_TIMFAULTENABLE_NONE

No fault enabled

HRTIM_TIMFAULTENABLE_FAULT1

Fault 1 enabled

HRTIM_TIMFAULTENABLE_FAULT2

Fault 2 enabled

HRTIM_TIMFAULTENABLE_FAULT3

Fault 3 enabled

HRTIM_TIMFAULTENABLE_FAULT4

Fault 4 enabled

HRTIM_TIMFAULTENABLE_FAULT5

Fault 5 enabled

HRTIM_TIMFAULTENABLE_FAULT6

Fault 6 enabled

HRTIM Timer Fault Lock

HRTIM_TIMFAULTLOCK_READWRITE

Timer fault enabling bits are read/write

HRTIM_TIMFAULTLOCK_READONLY

Timer fault enabling bits are read only

HRTIM Timer Greater than Compare 1 PWM Mode

HRTIM_TIMERGTCMP1_EQUAL

Timer Compare 1 event is generated when counter is equal

HRTIM_TIMERGTCMP1_GREATER

Timer Compare 1 event is generated when counter is greater

HRTIM Timer Greater than Compare 3 PWM Mode

HRTIM_TIMERGTCMP3_EQUAL

Timer Compare 3 event is generated when counter is equal

HRTIM_TIMERGTCMP3_GREATER

Timer Compare 3 Reset event is generated when counter is greater

HRTIM Timer identifier

HRTIM_TIMERID_MASTER

Master identifier

HRTIM_TIMERID_TIMER_A

Timer A identifier

HRTIM_TIMERID_TIMER_B

Timer B identifier

HRTIM_TIMERID_TIMER_C

Timer C identifier

HRTIM_TIMERID_TIMER_D

Timer D identifier

HRTIM_TIMERID_TIMER_E

Timer E identifier

HRTIM_TIMERID_TIMER_F

Timer F identifier

HRTIM Timer Index

HRTIM_TIMERINDEX_TIMER_A

Index used to access timer A registers

HRTIM_TIMERINDEX_TIMER_B

Index used to access timer B registers

HRTIM_TIMERINDEX_TIMER_C

Index used to access timer C registers

HRTIM_TIMERINDEX_TIMER_D

Index used to access timer D registers

HRTIM_TIMERINDEX_TIMER_E

Index used to access timer E registers

HRTIM_TIMERINDEX_TIMER_F

Index used to access timer F registers

HRTIM_TIMERINDEX_MASTER

Index used to access master registers

HRTIM_TIMERINDEX_COMMON

Index used to access HRTIM common registers

HRTIM Timer Output

HRTIM_OUTPUT_TA1

Timer A - Output 1 identifier

HRTIM_OUTPUT_TA2

Timer A - Output 2 identifier

HRTIM_OUTPUT_TB1

Timer B - Output 1 identifier

HRTIM_OUTPUT_TB2

Timer B - Output 2 identifier

HRTIM_OUTPUT_TC1

Timer C - Output 1 identifier

HRTIM_OUTPUT_TC2

Timer C - Output 2 identifier

HRTIM_OUTPUT_TD1

Timer D - Output 1 identifier

HRTIM_OUTPUT_TD2

Timer D - Output 2 identifier

HRTIM_OUTPUT_TE1

Timer E - Output 1 identifier

HRTIM_OUTPUT_TE2

Timer E - Output 2 identifier

HRTIM_OUTPUT_TF1

Timer F - Output 1 identifier

HRTIM_OUTPUT_TF2

Timer F - Output 2 identifier

HRTIM Timer Push Pull Mode

HRTIM_TIMPUSHPULLMODE_DISABLED

Push-Pull mode disabled

HRTIM_TIMPUSHPULLMODE_ENABLED

Push-Pull mode enabled

HRTIM Timer Repetition Update

HRTIM_UPDATEONREPETITION_DISABLED

Update on repetition disabled

HRTIM_UPDATEONREPETITION_ENABLED

Update on repetition enabled

HRTIM Timer Reset Trigger

HRTIM_TIMRESETTRIGGER_NONE

No counter reset trigger

HRTIM_TIMRESETTRIGGER_UPDATE

The timer counter is reset upon update event

HRTIM_TIMRESETTRIGGER_CMP2

The timer counter is reset upon Timer Compare 2 event

HRTIM_TIMRESETTRIGGER_CMP4

The timer counter is reset upon Timer Compare 4 event

HRTIM_TIMRESETTRIGGER_MASTER_PER

The timer counter is reset upon master timer period event

HRTIM_TIMRESETTRIGGER_MASTER_CMP1

The timer counter is reset upon master timer Compare 1 event

HRTIM_TIMRESETTRIGGER_MASTER_CMP2

The timer counter is reset upon master timer Compare 2 event

HRTIM_TIMRESETTRIGGER_MASTER_CMP3

The timer counter is reset upon master timer Compare 3 event

HRTIM_TIMRESETTRIGGER_MASTER_CMP4

The timer counter is reset upon master timer Compare 4 event

HRTIM_TIMRESETTRIGGER_EEV_1

The timer counter is reset upon external event 1U

HRTIM_TIMRESETTRIGGER_EEV_2

The timer counter is reset upon external event 2U

HRTIM_TIMRESETTRIGGER_EEV_3

The timer counter is reset upon external event 3U

HRTIM_TIMRESETTRIGGER_EEV_4

The timer counter is reset upon external event 4U

HRTIM_TIMRESETTRIGGER_EEV_5

The timer counter is reset upon external event 5U

HRTIM_TIMRESETTRIGGER_EEV_6

The timer counter is reset upon external event 6U

HRTIM_TIMRESETTRIGGER_EEV_7

The timer counter is reset upon external event 7U

HRTIM_TIMRESETTRIGGER_EEV_8

The timer counter is reset upon external event 8U

HRTIM_TIMRESETTRIGGER_EEV_9

The timer counter is reset upon external event 9U

HRTIM_TIMRESETTRIGGER_EEV_10

The timer counter is reset upon external event 10U

HRTIM_TIMRESETTRIGGER_OTHER1_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER1_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER1_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER2_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER2_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER2_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER3_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER3_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER3_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER4_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER4_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER4_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER5_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER5_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM Timer Reset Update

HRTIM_TIMUPDATEONRESET_DISABLED

Update by timer x reset / roll-over disabled

HRTIM_TIMUPDATEONRESET_ENABLED

Update by timer x reset / roll-over enabled

HRTIM Timer Re-Synchronized update

HRTIM_TIMERESYNC_UPDATE_UNCONDITIONAL

update taken into account immediately

HRTIM_TIMERESYNC_UPDATE_CONDITIONAL

update taken into account on the following Reset/Roll-over event

HRTIM Timer RollOver Mode

HRTIM_TIM_FEROM_BOTH

Roll-over event used by

HRTIM_TIM_FEROM_CREST

the Fault and

HRTIM_TIM_FEROM_VALLEY

Event counters

HRTIM_TIM_BMROM_BOTH

Roll-over event used in the Burst mode controller

HRTIM_TIM_BMROM_CREST

as clock

HRTIM_TIM_BMROM_VALLEY

and as burst mode trigger

HRTIM_TIM_ADROM_BOTH

Roll-over event which triggers

HRTIM_TIM_ADROM_CREST

the

HRTIM_TIM_ADROM_VALLEY

ADC

HRTIM_TIM_OUTROM_BOTH

Roll-over event which sets and/or resets the ouputs

HRTIM_TIM_OUTROM_CREST

as per HRTIM_SETxyR

HRTIM_TIM_OUTROM_VALLEY

and HRTIM_RSTxyR settings

HRTIM_TIM_ROM_BOTH

Roll-over event with the following destinations: IRQ and DMA requests,

HRTIM_TIM_ROM_CREST

Update trigger (to transfer content from preload to active registers),

HRTIM_TIM_ROM_VALLEY

repetition counter decrement and External Event filtering

IS_HRTIM_ROLLOVERMODE

HRTIM Re-Synchronized Update

HRTIM_RSYNCUPDATE_DISABLE

The update is taken into account immediately

HRTIM_RSYNCUPDATE_ENABLE

The update is taken into account on the following Reset/Roll-over event.

HRTIM Timer Triggered-Half Mode

HRTIM_TIMERTRIGHALF_DISABLED

Timer Compare 2 register is behaving in standard mode

HRTIM_TIMERTRIGHALF_ENABLED

Timer Compare 2 register is behaving in triggered-half mode

LL_HRTIM_TRIGHALF_DISABLED

Timer Compare 2 register is behaving in standard mode

LL_HRTIM_TRIGHALF_ENABLED

Timer Compare 2 register is behaving in triggered-half mode

HRTIM Timer Update Trigger

HRTIM_TIMUPDATETRIGGER_NONE

Register update is disabled

HRTIM_TIMUPDATETRIGGER_MASTER

Register update is triggered by the master timer update

HRTIM_TIMUPDATETRIGGER_TIMER_A

Register update is triggered by the timer A update

HRTIM_TIMUPDATETRIGGER_TIMER_B

Register update is triggered by the timer B update

HRTIM_TIMUPDATETRIGGER_TIMER_C

Register update is triggered by the timer C update

HRTIM_TIMUPDATETRIGGER_TIMER_D

Register update is triggered by the timer D update

HRTIM_TIMUPDATETRIGGER_TIMER_E

Register update is triggered by the timer E update

HRTIM_TIMUPDATETRIGGER_TIMER_F

Register update is triggered by the timer F update

HRTIM Timer UpDown Mode

HRTIM_TIMERUPDOWNMODE_UP

Timer counter is operating in up-counting mode

HRTIM_TIMERUPDOWNMODE_UPDOWN

Timer counter is operating in up-down counting mode

HRTIM Timing Unit DMA Request Enable

HRTIM_TIM_DMA_NONE

No DMA request enable

HRTIM_TIM_DMA_CMP1

Timer compare 1 DMA request enable

HRTIM_TIM_DMA_CMP2

Timer compare 2 DMA request enable

HRTIM_TIM_DMA_CMP3

Timer compare 3 DMA request enable

HRTIM_TIM_DMA_CMP4

Timer compare 4 DMA request enable

HRTIM_TIM_DMA_REP

Timer repetition DMA request enable

HRTIM_TIM_DMA_UPD

Timer update DMA request enable

HRTIM_TIM_DMA_CPT1

Timer capture 1 DMA request enable

HRTIM_TIM_DMA_CPT2

Timer capture 2 DMA request enable

HRTIM_TIM_DMA_SET1

Timer output 1 set DMA request enable

HRTIM_TIM_DMA_RST1

Timer output 1 reset DMA request enable

HRTIM_TIM_DMA_SET2

Timer output 2 set DMA request enable

HRTIM_TIM_DMA_RST2

Timer output 2 reset DMA request enable

HRTIM_TIM_DMA_RST

Timer reset DMA request enable

HRTIM_TIM_DMA_DLYPRT

Timer delay protection DMA request enable

HRTIM Timing Unit Interrupt Enable

HRTIM_TIM_IT_NONE

No interrupt enabled

HRTIM_TIM_IT_CMP1

Timer compare 1 interrupt enable

HRTIM_TIM_IT_CMP2

Timer compare 2 interrupt enable

HRTIM_TIM_IT_CMP3

Timer compare 3 interrupt enable

HRTIM_TIM_IT_CMP4

Timer compare 4 interrupt enable

HRTIM_TIM_IT_REP

Timer repetition interrupt enable

HRTIM_TIM_IT_UPD

Timer update interrupt enable

HRTIM_TIM_IT_CPT1

Timer capture 1 interrupt enable

HRTIM_TIM_IT_CPT2

Timer capture 2 interrupt enable

HRTIM_TIM_IT_SET1

Timer output 1 set interrupt enable

HRTIM_TIM_IT_RST1

Timer output 1 reset interrupt enable

HRTIM_TIM_IT_SET2

Timer output 2 set interrupt enable

HRTIM_TIM_IT_RST2

Timer output 2 reset interrupt enable

HRTIM_TIM_IT_RST

Timer reset interrupt enable

HRTIM_TIM_IT_DLYPRT

Timer delay protection interrupt enable

HRTIM Timing Unit Interrupt Flag

HRTIM_TIM_FLAG_CMP1

Timer compare 1 interrupt flag

HRTIM_TIM_FLAG_CMP2

Timer compare 2 interrupt flag

HRTIM_TIM_FLAG_CMP3

Timer compare 3 interrupt flag

HRTIM_TIM_FLAG_CMP4

Timer compare 4 interrupt flag

HRTIM_TIM_FLAG_REP

Timer repetition interrupt flag

HRTIM_TIM_FLAG_UPD

Timer update interrupt flag

HRTIM_TIM_FLAG_CPT1

Timer capture 1 interrupt flag

HRTIM_TIM_FLAG_CPT2

Timer capture 2 interrupt flag

HRTIM_TIM_FLAG_SET1

Timer output 1 set interrupt flag

HRTIM_TIM_FLAG_RST1

Timer output 1 reset interrupt flag

HRTIM_TIM_FLAG_SET2

Timer output 2 set interrupt flag

HRTIM_TIM_FLAG_RST2

Timer output 2 reset interrupt flag

HRTIM_TIM_FLAG_RST

Timer reset interrupt flag

HRTIM_TIM_FLAG_DLYPRT

Timer delay protection interrupt flag

HRTIM Update Gating

HRTIM_UPDATEGATING_INDEPENDENT

Update done independently from the DMA burst transfer completion

HRTIM_UPDATEGATING_DMABURST

Update done when the DMA burst transfer is completed

HRTIM_UPDATEGATING_DMABURST_UPDATE

Update done on timer roll-over following a DMA burst transfer completion

HRTIM_UPDATEGATING_UPDEN1

Slave timer only - Update done on a rising edge of HRTIM update enable input 1U

HRTIM_UPDATEGATING_UPDEN2

Slave timer only - Update done on a rising edge of HRTIM update enable input 2U

HRTIM_UPDATEGATING_UPDEN3

Slave timer only - Update done on a rising edge of HRTIM update enable input 3U

HRTIM_UPDATEGATING_UPDEN1_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 1U

HRTIM_UPDATEGATING_UPDEN2_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 2U

HRTIM_UPDATEGATING_UPDEN3_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 3U

29 HAL I2C Generic Driver

29.1 I2C Firmware driver registers structures

29.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32g4xx_hal_i2c.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***

  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::OwnAddress1***

  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***

  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***

  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_DUAL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***

  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***

  Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***I2C_OWN_ADDRESS2_MASKS***
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***

  Specifies if general call mode is selected. This parameter can be a value of ***I2C_GENERAL_CALL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***

  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_NOSTRETCH_MODE***

29.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32g4xx_hal_i2c.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***

- ***__IO uint32_t PreviousState***
- ***HAL_StatusTypeDef(* XferISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO HAL_I2C_ModeTypeDef Mode***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t AddrEventCount***

Field Documentation

- ***I2C_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequantial transfer options, this parameter can be a value of *I2C_XFEROPTIONS*
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter

29.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

29.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;

2. Initialize the I2C low level resources by implementing the @ref HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock
   b. I2C pins configuration
      - Enable the clock for the I2C GPIOs
      - Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the I2Cx interrupt priority
      - Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
      - Enable the DMAx interface clock using
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx channel
      - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the @ref HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized @ref HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function @ref HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using @ref HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using @ref HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using @ref HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using @ref HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using @ref HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using @ref HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using @ref HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, @ref HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using @ref HAL_I2C_Master_Receive_IT()
- At reception end of transfer, @ref HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using @ref HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, @ref HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using @ref HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, @ref HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, @ref HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ErrorCallback()

- Abort a master I2C process communication with Interrupt using @ref HAL_I2C_Master_Abort_IT()
- End of abort process, @ref HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using @ref __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode or DMA mode IO sequential operation

Note: These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer
- Option field values are defined through @ref I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functionnal is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like @ref HAL_I2C_Master_Seq_Transmit_IT() then @ref HAL_I2C_Master_Seq_Transmit_IT() or @ref HAL_I2C_Master_Seq_Transmit_DMA() then @ref HAL_I2C_Master_Seq_Transmit_DMA())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  - I2C_LAST_FRAME_NO_STOP: Sequential usage (Master only), this option allow to manage a restart condition after several call of the same master sequential interface several times (link with option I2C_FIRST_AND_NEXT_FRAME). Usage can, transfer several bytes one by one using HAL_I2C_Master_Seq_Transmit_IT(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME) or HAL_I2C_Master_Seq_Receive_IT(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME) or HAL_I2C_Master_Seq_Transmit_DMA(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME) or HAL_I2C_Master_Seq_Receive_DMA(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME). Then usage of this option I2C_LAST_FRAME_NO_STOP at the last Transmit or Receive sequence permit to call the oposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - I2C_OTHER_FRAME: Sequential usage (Master only), this option allow to manage a restart condition after each call of the same master sequential interface. Usage can, transfer several bytes one by one with a restart with slave address between each bytes using HAL_I2C_Master_Seq_Transmit_IT(option I2C_FIRST_FRAME then I2C_OTHER_FRAME) or HAL_I2C_Master_Seq_Receive_IT(option I2C_FIRST_FRAME then I2C_OTHER_FRAME) or HAL_I2C_Master_Seq_Transmit_DMA(option I2C_FIRST_FRAME then I2C_OTHER_FRAME) or HAL_I2C_Master_Seq_Receive_DMA(option I2C_FIRST_FRAME then I2C_OTHER_FRAME). Then usage of this option I2C_OTHER_AND_LAST_FRAME at the last frame to help automatic generation of STOP condition.

- Differents sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Master_Seq_Transmit_IT() or using @ref HAL_I2C_Master_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, @ref HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Master_Seq_Receive_IT() or using @ref HAL_I2C_Master_Seq_Receive_DMA()
    - At reception end of current frame transfer, @ref HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterRxCpltCallback()
  - Abort a master IT or DMA I2C process communication with Interrupt using @ref HAL_I2C_Master_Abort_IT()
    - End of abort process, @ref HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using @ref HAL_I2C_EnableListen_IT() @ref HAL_I2C_DisableListen_IT()
    - When address slave I2C match, @ref HAL_I2C_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end @ref HAL_I2C_ListenCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Slave_Seq_Transmit_IT() or using @ref HAL_I2C_Slave_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, @ref HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Slave_Seq_Receive_IT() or using @ref HAL_I2C_Slave_Seq_Receive_DMA()
    - At reception end of current frame transfer, @ref HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, @ref HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ErrorCallback()
  - Discard a slave I2C process communication using @ref __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using @ref HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, @ref HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using @ref HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, @ref HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, @ref HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using @ref HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, @ref HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterTxCpltCallback()

- Receive in master mode an amount of data in non-blocking mode (DMA) using @ref HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, @ref HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using @ref HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, @ref HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using @ref HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, @ref HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, @ref HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using @ref HAL_I2C_Master_Abort_IT()
- End of abort process, @ref HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using @ref __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using @ref HAL_I2C_Mem_Write_DMA()
- At Memory end of write transfer, @ref HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using @ref HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, @ref HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, @ref HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- @ref __HAL_I2C_ENABLE: Enable the I2C peripheral
- @ref __HAL_I2C_DISABLE: Disable the I2C peripheral
- @ref __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- @ref __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- @ref __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- @ref __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- @ref __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

Callback registration

The compilation flag USE_HAL_I2C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_I2C_RegisterCallback() or @ref HAL_I2C_RegisterAddrCallback() to register an interrupt callback.

Function @ref HAL_I2C_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.

- MemTxCpltCallback : callback for Memory transmission end of transfer.
- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : @ref HAL_I2C_RegisterAddrCallback().

Use function @ref HAL_I2C_UnRegisterCallback to reset a callback to the default weak function. @ref HAL_I2C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.
- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : @ref HAL_I2C_UnRegisterAddrCallback().

By default, after the @ref HAL_I2C_Init() and when the state is @ref HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples @ref HAL_I2C_MasterTxCpltCallback(), @ref HAL_I2C_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_I2C_Init()/ @ref HAL_I2C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the @ref HAL_I2C_Init()/ @ref HAL_I2C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in @ref HAL_I2C_STATE_READY or @ref HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_I2C_RegisterCallback() before calling @ref HAL_I2C_DeInit() or @ref HAL_I2C_Init() function.

When the compilation flag USE_HAL_I2C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I2C HAL driver header file for more useful macros

29.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode

- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- ***HAL_I2C_Init***
- ***HAL_I2C_DeInit***
- ***HAL_I2C_MspInit***
- ***HAL_I2C_MspDeInit***

29.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2C_Master_Transmit()
   - HAL_I2C_Master_Receive()
   - HAL_I2C_Slave_Transmit()
   - HAL_I2C_Slave_Receive()
   - HAL_I2C_Mem_Write()
   - HAL_I2C_Mem_Read()
   - HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2C_Master_Transmit_IT()
   - HAL_I2C_Master_Receive_IT()
   - HAL_I2C_Slave_Transmit_IT()
   - HAL_I2C_Slave_Receive_IT()
   - HAL_I2C_Mem_Write_IT()
   - HAL_I2C_Mem_Read_IT()
   - HAL_I2C_Master_Seq_Transmit_IT()
   - HAL_I2C_Master_Seq_Receive_IT()
   - HAL_I2C_Slave_Seq_Transmit_IT()
   - HAL_I2C_Slave_Seq_Receive_IT()
   - HAL_I2C_EnableListen_IT()
   - HAL_I2C_DisableListen_IT()
   - HAL_I2C_Master_Abort_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2C_Master_Transmit_DMA()
   - HAL_I2C_Master_Receive_DMA()
   - HAL_I2C_Slave_Transmit_DMA()
   - HAL_I2C_Slave_Receive_DMA()
   - HAL_I2C_Mem_Write_DMA()
   - HAL_I2C_Mem_Read_DMA()
   - HAL_I2C_Master_Seq_Transmit_DMA()
   - HAL_I2C_Master_Seq_Receive_DMA()
   - HAL_I2C_Slave_Seq_Transmit_DMA()
   - HAL_I2C_Slave_Seq_Receive_DMA()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2C_MasterTxCpltCallback()
   - HAL_I2C_MasterRxCpltCallback()
   - HAL_I2C_SlaveTxCpltCallback()
   - HAL_I2C_SlaveRxCpltCallback()
   - HAL_I2C_MemTxCpltCallback()
   - HAL_I2C_MemRxCpltCallback()
   - HAL_I2C_AddrCallback()
   - HAL_I2C_ListenCpltCallback()
   - HAL_I2C_ErrorCallback()
   - HAL_I2C_AbortCpltCallback()

This section contains the following APIs:

- ***HAL_I2C_Master_Transmit***
- ***HAL_I2C_Master_Receive***
- ***HAL_I2C_Slave_Transmit***
- ***HAL_I2C_Slave_Receive***
- ***HAL_I2C_Master_Transmit_IT***
- ***HAL_I2C_Master_Receive_IT***
- ***HAL_I2C_Slave_Transmit_IT***
- ***HAL_I2C_Slave_Receive_IT***
- ***HAL_I2C_Master_Transmit_DMA***
- ***HAL_I2C_Master_Receive_DMA***
- ***HAL_I2C_Slave_Transmit_DMA***
- ***HAL_I2C_Slave_Receive_DMA***
- ***HAL_I2C_Mem_Write***
- ***HAL_I2C_Mem_Read***
- ***HAL_I2C_Mem_Write_IT***
- ***HAL_I2C_Mem_Read_IT***
- ***HAL_I2C_Mem_Write_DMA***
- ***HAL_I2C_Mem_Read_DMA***
- ***HAL_I2C_IsDeviceReady***
- ***HAL_I2C_Master_Seq_Transmit_IT***
- ***HAL_I2C_Master_Seq_Transmit_DMA***
- ***HAL_I2C_Master_Seq_Receive_IT***
- ***HAL_I2C_Master_Seq_Receive_DMA***
- ***HAL_I2C_Slave_Seq_Transmit_IT***
- ***HAL_I2C_Slave_Seq_Transmit_DMA***
- ***HAL_I2C_Slave_Seq_Receive_IT***
- ***HAL_I2C_Slave_Seq_Receive_DMA***
- ***HAL_I2C_EnableListen_IT***
- ***HAL_I2C_DisableListen_IT***
- ***HAL_I2C_Master_Abort_IT***

29.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_I2C_GetState***
- ***HAL_I2C_GetMode***
- ***HAL_I2C_GetError***

29.2.5 Detailed description of functions

HAL_I2C_Init

Function name

HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)

Function description

Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DeInit

Function name

HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C peripheral.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

Initialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MspDeInit

Function name

void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_Master_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Write

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Write an amount of data in blocking mode to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Read

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Read an amount of data in blocking mode from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Checks if target device is ready for communication.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with Interrupt to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C

Return values

- **HAL:** status

HAL_I2C_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)

Function description

Abort a master I2C IT or DMA process communication with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with DMA to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Reads an amount of data in non-blocking mode with DMA from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be read

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EV_IRQHandler

Function name

void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C event interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ER_IRQHandler

Function name

void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C error interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterTxCpltCallback

Function name

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterRxCpltCallback

Function name

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveTxCpltCallback

Function name

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveRxCpltCallback

Function name

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AddrCallback

Function name

void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of I2C Transfer Direction Master Point of View
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_I2C_ListenCpltCallback

Function name

void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Listen Complete callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemTxCpltCallback

Function name

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemRxCpltCallback

Function name

void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ErrorCallback

Function name

void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C error callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AbortCpltCallback

Function name

void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C abort callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_GetState

Function name

HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)

Function description

Return the I2C handle state.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** state

HAL_I2C_GetMode

Function name

HAL_I2C_ModeTypeDef HAL_I2C_GetMode (I2C_HandleTypeDef * hi2c)

Function description

Returns the I2C Master, Slave, Memory or no mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values

- **HAL:** mode

HAL_I2C_GetError

Function name

uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)

Function description

Return the I2C error code.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **I2C:** Error Code

29.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

29.3.1 I2C

I2C

I2C Addressing Mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C Dual Addressing Mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code definition

HAL_I2C_ERROR_NONE

No error

HAL_I2C_ERROR_BERR

BERR error

HAL_I2C_ERROR_ARLO

ARLO error

HAL_I2C_ERROR_AF

ACKF error

HAL_I2C_ERROR_OVR

OVR error

HAL_I2C_ERROR_DMA

DMA transfer error

HAL_I2C_ERROR_TIMEOUT

Timeout error

HAL_I2C_ERROR_SIZE

Size Management error

HAL_I2C_ERROR_DMA_PARAM

DMA Parameter Error

HAL_I2C_ERROR_INVALID_PARAM

Invalid Parameters error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

I2C_FLAG_MASK

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_TXIS Transmit interrupt status
  - I2C_FLAG_RXNE Receive data register not empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_TC Transfer complete (master mode)
  - I2C_FLAG_TCR Transfer complete reload
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert
  - I2C_FLAG_BUSY Bus busy
  - I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I2C_GET_FLAG

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_GENERATE_NACK

Description:

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

I2C_LAST_FRAME_NO_STOP

I2C_OTHER_FRAME

I2C_OTHER_AND_LAST_FRAME

30 HAL I2C Extension Driver

30.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

30.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32G4xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus

30.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
   - HAL_I2CEx_EnableWakeUp()
   - HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions :
   - HAL_I2CEx_EnableFastModePlus()
   - HAL_I2CEx_DisableFastModePlus()

30.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Wake Up Feature
- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_I2CEx_ConfigAnalogFilter***
- ***HAL_I2CEx_ConfigDigitalFilter***
- ***HAL_I2CEx_EnableWakeUp***
- ***HAL_I2CEx_DisableWakeUp***
- ***HAL_I2CEx_EnableFastModePlus***
- ***HAL_I2CEx_DisableFastModePlus***

30.1.4 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)

Function description

Configure I2C Analog noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **AnalogFilter:** New state of the Analog filter.

Return values

- **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)

Function description

Configure I2C Digital noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Enable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Disable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_EnableFastModePlus

Function name

void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C4 parameter.

HAL_I2CEx_DisableFastModePlus

Function name

void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C4 parameter.

30.2 I2CEx Firmware driver defines

The following section lists the various define and macros of the module.

30.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FMP_NOT_SUPPORTED

Fast Mode Plus not supported

I2C_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

I2C_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

I2C_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

I2C_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

I2C_FASTMODEPLUS_I2C1

Enable Fast Mode Plus on I2C1 pins

I2C_FASTMODEPLUS_I2C2

Enable Fast Mode Plus on I2C2 pins

I2C_FASTMODEPLUS_I2C3

Enable Fast Mode Plus on I2C3 pins

I2C_FASTMODEPLUS_I2C4

Enable Fast Mode Plus on I2C4 pins

31 HAL I2S Generic Driver

31.1 I2S Firmware driver registers structures

31.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32g4xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***

Field Documentation

- ***uint32_t I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint32_t I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint32_t I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint32_t I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint32_t I2S_InitTypeDef::CPOL***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***

31.1.2 I2S_HandleTypeDef

I2S_HandleTypeDef is defined in the stm32g4xx_hal_i2s.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***I2S_InitTypeDef Init***
- ***uint16_t * pTxBuffPtr***
- ***__IO uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint16_t * pRxBuffPtr***
- ***__IO uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_I2S_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SPI_TypeDef* I2S_HandleTypeDef::Instance***
  I2S registers base address
- ***I2S_InitTypeDef I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***uint16_t* I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters
- ***__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO uint32_t I2S_HandleTypeDef::ErrorCode***
  I2S Error code This parameter can be a value of ***I2S_Error***

31.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

31.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.

2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      - Enable the clock for the I2S GPIOs.
      - Configure these I2S pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
      - Configure the I2Sx interrupt priority.
      - Enable the NVIC I2S IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs:
      - Declare a DMA handle structure for the Tx/Rx Stream/Channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream/Channel.
      - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream/Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function.

Note: The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that either:

– *SYSCLK is configured or*
– *PLLADCCLK output is configured or*
– *HSI is enabled or*
– *External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32g4xx_hal_conf.h file.*

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()

- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not

Note: You can refer to the I2S HAL driver header file for more useful macros

I2S HAL driver macros list

Callback registration:

1. The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1U allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback. Function HAL_I2S_RegisterCallback() allows to register following callbacks:
   - TxCpltCallback : I2S Tx Completed callback
   - RxCpltCallback : I2S Rx Completed callback
   - TxHalfCpltCallback : I2S Tx Half Completed callback
   - RxHalfCpltCallback : I2S Rx Half Completed callback
   - ErrorCallback : I2S Error callback
   - MspInitCallback : I2S Msp Init callback
   - MspDeInitCallback : I2S Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   - TxCpltCallback : I2S Tx Completed callback
   - RxCpltCallback : I2S Rx Completed callback
   - TxHalfCpltCallback : I2S Tx Half Completed callback
   - RxHalfCpltCallback : I2S Rx Half Completed callback
   - ErrorCallback : I2S Error callback
   - MspInitCallback : I2S Msp Init callback
   - MspDeInitCallback : I2S Msp DeInit callback

By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function.

When the compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

31.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function HAL_I2S_DeInit() to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- ***HAL_I2S_Init***
- ***HAL_I2S_DeInit***
- ***HAL_I2S_MspInit***
- ***HAL_I2S_MspDeInit***

31.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2S_Transmit()
   - HAL_I2S_Receive()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2S_Transmit_IT()
   - HAL_I2S_Receive_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2S_Transmit_DMA()
   - HAL_I2S_Receive_DMA()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   – HAL_I2S_TxCpltCallback()
   – HAL_I2S_RxCpltCallback()
   – HAL_I2S_ErrorCallback()

This section contains the following APIs:

- ***HAL_I2S_Transmit***
- ***HAL_I2S_Receive***
- ***HAL_I2S_Transmit_IT***
- ***HAL_I2S_Receive_IT***
- ***HAL_I2S_Transmit_DMA***
- ***HAL_I2S_Receive_DMA***
- ***HAL_I2S_DMAPause***
- ***HAL_I2S_DMAResume***
- ***HAL_I2S_DMAStop***
- ***HAL_I2S_IRQHandler***
- ***HAL_I2S_TxHalfCpltCallback***
- ***HAL_I2S_TxCpltCallback***
- ***HAL_I2S_RxHalfCpltCallback***
- ***HAL_I2S_RxCpltCallback***
- ***HAL_I2S_ErrorCallback***

31.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_I2S_GetState***
- ***HAL_I2S_GetError***

31.2.5 Detailed description of functions

HAL_I2S_Init

Function name

HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)

Function description

Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DeInit

Function name

HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)

Function description

DeInitializes the I2S peripheral.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_MspInit

Function name

void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP Init.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_MspDeInit

Function name

void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP DeInit.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit

Function name

HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive

Function name

HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.

HAL_I2S_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronization between Master and Slave otherwise the I2S interrupt should be optimized.

HAL_I2S_IRQHandler

Function name

void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)

Function description

This function handles I2S interrupt request.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Transmit data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_DMAPause

Function name

HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)

Function description

Pauses the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAResume

Function name

HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)

Function description

Resumes the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAStop

Function name

HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)

Function description

Stops the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_TxHalfCpltCallback

Function name

void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer Half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_TxCpltCallback

Function name

void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxHalfCpltCallback

Function name

void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxCpltCallback

Function name

void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_ErrorCallback

Function name

void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)

Function description

I2S error callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_GetState

Function name

HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)

Function description

Return the I2S state.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** state

HAL_I2S_GetError

Function name

uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)

Function description

Return the I2S error code.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **I2S:** Error Code

31.3 I2S Firmware driver defines

The following section lists the various define and macros of the module.

31.3.1 I2S

I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error

HAL_I2S_ERROR_NONE

No error

HAL_I2S_ERROR_TIMEOUT

Timeout error

HAL_I2S_ERROR_OVR

OVR error

HAL_I2S_ERROR_UDR

UDR error

HAL_I2S_ERROR_DMA

DMA transfer error

HAL_I2S_ERROR_PRESCALER

Prescaler Calculation error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_TXE: Tx buffer empty interrupt enable
  - I2S_IT_RXNE: RX buffer not empty interrupt enable
  - I2S_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_I2S_DISABLE_IT

Description:

- Disable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_TXE: Tx buffer empty interrupt enable
  - I2S_IT_RXNE: RX buffer not empty interrupt enable
  - I2S_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_I2S_GET_IT_SOURCE

Description:

- Checks if the specified I2S interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - I2S_IT_TXE: Tx buffer empty interrupt enable
  - I2S_IT_RXNE: RX buffer not empty interrupt enable
  - I2S_IT_ERR: Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_I2S_GET_FLAG

Description:

- Checks whether the specified I2S flag is set or not.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2S_FLAG_RXNE: Receive buffer not empty flag
  - I2S_FLAG_TXE: Transmit buffer empty flag
  - I2S_FLAG_UDR: Underrun flag
  - I2S_FLAG_OVR: Overrun flag
  - I2S_FLAG_FRE: Frame error flag
  - I2S_FLAG_CHSIDE: Channel Side flag
  - I2S_FLAG_BSY: Busy flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clears the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clears the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

I2S Flags Definition

I2S_FLAG_TXE

I2S_FLAG_RXNE

I2S_FLAG_UDR

I2S_FLAG_OVR

I2S_FLAG_FRE

I2S_FLAG_CHSIDE

I2S_FLAG_BSY

I2S_FLAG_MASK

I2S Interrupts Definition

I2S_IT_TXE

I2S_IT_RXNE

I2S_IT_ERR

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

32 HAL IRDA Generic Driver

32.1 IRDA Firmware driver registers structures

32.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32g4xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***

  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hirda->Init.BaudRate))) where usart_ker_ckpres is the IRDA input clock divided by a prescaler
- ***uint32_t IRDA_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDAEx_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***

  Note:

  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***

  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.

  Note:

  – Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***

  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***
- ***uint32_t IRDA_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the IRDA clock source. This parameter can be a value of ***IRDA_ClockPrescaler***.

32.1.2 IRDA_HandleTypeDef

IRDA_HandleTypeDef is defined in the stm32g4xx_hal_irda.h

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***

- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***uint16_t IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO uint32_t IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code

32.2 IRDA Firmware driver API description

The following section lists the various functions of the IRDA library.

32.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
   - Enable the USARTx/UARTx interface clock.
   - USARTx/UARTx pins configuration:
     - Enable the clock for the USARTx/UARTx GPIOs.
     - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
     - Configure the USARTx/UARTx interrupt priority.
     - Enable the NVIC USARTx/UARTx IRQ handle.
     - The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
   - DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.
4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API.

Note: The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.

5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()

- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

Note: You can refer to the IRDA HAL driver header file for more useful macros

32.2.2 Callback registration

The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_IRDA_RegisterCallback() to register a user callback. Function @ref HAL_IRDA_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_IRDA_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_IRDA_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit.

By default, after the @ref HAL_IRDA_Init() and when the state is HAL_IRDA_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples @ref HAL_IRDA_TxCpltCallback(), @ref HAL_IRDA_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_IRDA_Init() and @ref HAL_IRDA_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_IRDA_Init() and @ref HAL_IRDA_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_IRDA_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_IRDA_STATE_READY or HAL_IRDA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_IRDA_RegisterCallback() before calling @ref HAL_IRDA_DeInit() or @ref HAL_IRDA_Init() function.

When The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

32.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_IRDA_Init***
- ***HAL_IRDA_DeInit***
- ***HAL_IRDA_MspInit***
- ***HAL_IRDA_MspDeInit***

32.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   - Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - Non-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
   - HAL_IRDA_Transmit()
   - HAL_IRDA_Receive()

3. Non Blocking mode APIs with Interrupt are :
   - HAL_IRDA_Transmit_IT()
   - HAL_IRDA_Receive_IT()
   - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_IRDA_Transmit_DMA()
   - HAL_IRDA_Receive_DMA()
   - HAL_IRDA_DMAPause()
   - HAL_IRDA_DMAResume()
   - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
   - HAL_IRDA_TxHalfCpltCallback()
   - HAL_IRDA_TxCpltCallback()
   - HAL_IRDA_RxHalfCpltCallback()
   - HAL_IRDA_RxCpltCallback()
   - HAL_IRDA_ErrorCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   - HAL_IRDA_Abort()
   - HAL_IRDA_AbortTransmit()
   - HAL_IRDA_AbortReceive()
   - HAL_IRDA_Abort_IT()
   - HAL_IRDA_AbortTransmit_IT()
   - HAL_IRDA_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   - HAL_IRDA_AbortCpltCallback()
   - HAL_IRDA_AbortTransmitCpltCallback()
   - HAL_IRDA_AbortReceiveCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   - Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept ongoing on IRDA side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_IRDA_Transmit***
- ***HAL_IRDA_Receive***
- ***HAL_IRDA_Transmit_IT***
- ***HAL_IRDA_Receive_IT***
- ***HAL_IRDA_Transmit_DMA***
- ***HAL_IRDA_Receive_DMA***
- ***HAL_IRDA_DMAPause***
- ***HAL_IRDA_DMAResume***
- ***HAL_IRDA_DMAStop***
- ***HAL_IRDA_Abort***
- ***HAL_IRDA_AbortTransmit***
- ***HAL_IRDA_AbortReceive***

- *HAL_IRDA_Abort_IT*
- *HAL_IRDA_AbortTransmit_IT*
- *HAL_IRDA_AbortReceive_IT*
- *HAL_IRDA_IRQHandler*
- *HAL_IRDA_TxCpltCallback*
- *HAL_IRDA_TxHalfCpltCallback*
- *HAL_IRDA_RxCpltCallback*
- *HAL_IRDA_RxHalfCpltCallback*
- *HAL_IRDA_ErrorCallback*
- *HAL_IRDA_AbortCpltCallback*
- *HAL_IRDA_AbortTransmitCpltCallback*
- *HAL_IRDA_AbortReceiveCpltCallback*

32.2.5 Peripheral State and Error functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- *HAL_IRDA_GetState*
- *HAL_IRDA_GetError*

32.2.6 Detailed description of functions

HAL_IRDA_Init

Function name

HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DeInit

Function name

HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA peripheral.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_MspInit

Function name

void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_MspDeInit

Function name

void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_Transmit

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive

Function name

HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.
- When the IRDA parity is enabled (PCE = 1), the received data contains the parity bit (MSB position).

HAL_IRDA_DMAPause

Function name

HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)

Function description

Pause the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DMAResume

Function name

HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)

Function description

Resume the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_DMAStop

Function name

HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)

Function description

Stop the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_Abort

Function name

HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

Handle IRDA interrupt request.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxCpltCallback

Function name

void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RxCpltCallback

Function name

void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxHalfCpltCallback

Function name

void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Half Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- **None:**

HAL_IRDA_RxHalfCpltCallback

Function name

void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Half Transfer complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_ErrorCallback

Function name

void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA error callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortCpltCallback

Function name

void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name

void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name

void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Receive Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_GetState

Function name

HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle state.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** state

HAL_IRDA_GetError

Function name

uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle error code.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **IRDA:** Error Code

32.3 IRDA Firmware driver defines

The following section lists the various define and macros of the module.

32.3.1 IRDA

IRDA

IRDA Clock Prescaler

IRDA_PRESCALER_DIV1

fclk_pres = fclk

IRDA_PRESCALER_DIV2

fclk_pres = fclk/2

IRDA_PRESCALER_DIV4

fclk_pres = fclk/4

IRDA_PRESCALER_DIV6

fclk_pres = fclk/6

IRDA_PRESCALER_DIV8

fclk_pres = fclk/8

IRDA_PRESCALER_DIV10

fclk_pres = fclk/10

IRDA_PRESCALER_DIV12

fclk_pres = fclk/12

IRDA_PRESCALER_DIV16

fclk_pres = fclk/16

IRDA_PRESCALER_DIV32

fclk_pres = fclk/32

IRDA_PRESCALER_DIV64

fclk_pres = fclk/64

IRDA_PRESCALER_DIV128

fclk_pres = fclk/128

IRDA_PRESCALER_DIV256

fclk_pres = fclk/256

IRDA DMA Rx

IRDA_DMA_RX_DISABLE

IRDA DMA RX disabled

IRDA_DMA_RX_ENABLE

IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE

IRDA DMA TX disabled

IRDA_DMA_TX_ENABLE

IRDA DMA TX enabled

IRDA Error Code Definition

HAL_IRDA_ERROR_NONE

No error

HAL_IRDA_ERROR_PE

Parity error

HAL_IRDA_ERROR_NE

Noise error

HAL_IRDA_ERROR_FE

frame error

HAL_IRDA_ERROR_ORE

Overrun error

HAL_IRDA_ERROR_DMA

DMA transfer error

HAL_IRDA_ERROR_BUSY

Busy Error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_CLEAR_PEF
  - IRDA_CLEAR_FEF
  - IRDA_CLEAR_NEF
  - IRDA_CLEAR_OREF
  - IRDA_CLEAR_TCF
  - IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_GET_FLAG

Description:

- Check whether the specified IRDA flag is set or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_GET_IT

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ORE OverRun Error interrupt
  - IRDA_IT_NE Noise Error interrupt
  - IRDA_IT_FE Framing Error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_GET_IT_SOURCE

Description:

- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ERR Framing, overrun or noise error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_CLEAR_IT

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA_CLEAR_PEF Parity Error Clear Flag
  - IRDA_CLEAR_FEF Framing Error Clear Flag
  - IRDA_CLEAR_NEF Noise detected Clear Flag
  - IRDA_CLEAR_OREF OverRun Error Clear Flag
  - IRDA_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - IRDA_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - IRDA_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - IRDA_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

IRDA Flags

IRDA_FLAG_REACK

IRDA receive enable acknowledge flag

IRDA_FLAG_TEACK

IRDA transmit enable acknowledge flag

IRDA_FLAG_BUSY

IRDA busy flag

IRDA_FLAG_ABRF

IRDA auto Baud rate flag

IRDA_FLAG_ABRE

IRDA auto Baud rate error

IRDA_FLAG_TXE

IRDA transmit data register empty

IRDA_FLAG_TC

IRDA transmission complete

IRDA_FLAG_RXNE

IRDA read data register not empty

IRDA_FLAG_ORE

IRDA overrun error

IRDA_FLAG_NE

IRDA noise error

IRDA_FLAG_FE

IRDA frame error

IRDA_FLAG_PE

IRDA parity error

IRDA interruptions flags mask

IRDA_IT_MASK

IRDA Interruptions flags mask

IRDA_CR_MASK

IRDA control register mask

IRDA_CR_POS

IRDA control register position

IRDA_ISR_MASK

IRDA ISR register mask

IRDA_ISR_POS

IRDA ISR register position

IRDA Interrupts Definition

IRDA_IT_PE

IRDA Parity error interruption

IRDA_IT_TXE

IRDA Transmit data register empty interruption

IRDA_IT_TC

IRDA Transmission complete interruption

IRDA_IT_RXNE

IRDA Read data register not empty interruption

IRDA_IT_IDLE

IRDA Idle interruption

IRDA_IT_ERR

IRDA Error interruption

IRDA_IT_ORE

IRDA Overrun error interruption

IRDA_IT_NE

IRDA Noise error interruption

IRDA_IT_FE

IRDA Frame error interruption

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF

Parity Error Clear Flag

IRDA_CLEAR_FEF

Framing Error Clear Flag

IRDA_CLEAR_NEF

Noise Error detected Clear Flag

IRDA_CLEAR_OREF

OverRun Error Clear Flag

IRDA_CLEAR_IDLEF

IDLE line detected Clear Flag

IRDA_CLEAR_TCF

Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL

IRDA normal power mode

IRDA_POWERMODE_LOWPOWER

IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE

Associated UART disabled in IRDA mode

IRDA_MODE_ENABLE

Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disabled

IRDA_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE

No parity

IRDA_PARITY_EVEN

Even parity

IRDA_PARITY_ODD

Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST

Auto-Baud Rate Request

IRDA_RXDATA_FLUSH_REQUEST

Receive Data flush Request

IRDA_TXDATA_FLUSH_REQUEST

Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE

IRDA disabled

IRDA_STATE_ENABLE

IRDA enabled

IRDA State Code Definition

HAL_IRDA_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_IRDA_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_IRDA_STATE_BUSY

An internal process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_IRDA_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_IRDA_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_IRDA_STATE_ERROR

Error Value is allowed for gState only

IRDA Transfer Mode

IRDA_MODE_RX

RX mode

IRDA_MODE_TX

TX mode

IRDA_MODE_TX_RX

RX and TX mode

33 HAL IRDA Extension Driver

33.1 IRDAEx Firmware driver defines

The following section lists the various define and macros of the module.

33.1.1 IRDAEx

IRDAEx

IRDAEx Word Length

IRDA_WORDLENGTH_7B

7-bit long frame

IRDA_WORDLENGTH_8B

8-bit long frame

IRDA_WORDLENGTH_9B

9-bit long frame

34 HAL IWDG Generic Driver

34.1 IWDG Firmware driver registers structures

34.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32g4xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

34.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32g4xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
  IWDG required parameters

34.2 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

34.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by Low-Speed clock (LSI) and thus stays active even if the main clock fails.
- Once the IWDG is started, the LSI is forced ON and both can not be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode : When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros.

Min-max timeout value @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI frequency dispersion. STM32G4xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

34.2.2 How to use this driver

1. Use IWDG using HAL_IWDG_Init() function to :
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts counting down.
   - Enable write access to configuration registers: IWDG_PR, IWDG_RLR and IWDG_WINR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - Wait for status flags to be reset.
   - Depending on window parameter:
     - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function with exact time base.
     - Else modify Window register. This will automatically reload watchdog counter.
2. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

34.2.3 Initialization and Start functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Manage Window option.
- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:

- ***HAL_IWDG_Init***

34.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh***

34.2.5 Detailed description of functions

HAL_IWDG_Init

Function name

HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_Refresh

Function name

HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)

Function description

Refresh the IWDG.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

34.3 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

34.3.1 IWDG

IWDG

IWDG Exported Macros

__HAL_IWDG_START

Description:

- Enable the IWDG peripheral.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

IWDG Prescaler

IWDG_PRESCALER_4

IWDG prescaler set to 4

IWDG_PRESCALER_8

IWDG prescaler set to 8

IWDG_PRESCALER_16

IWDG prescaler set to 16

IWDG_PRESCALER_32

IWDG prescaler set to 32

IWDG_PRESCALER_64

IWDG prescaler set to 64

IWDG_PRESCALER_128

IWDG prescaler set to 128

IWDG_PRESCALER_256

IWDG prescaler set to 256

IWDG Window option

IWDG_WINDOW_DISABLE

35 HAL LPTIM Generic Driver

35.1 LPTIM Firmware driver registers structures

35.1.1 LPTIM_ClockConfigTypeDef

LPTIM_ClockConfigTypeDef is defined in the stm32g4xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

35.1.2 LPTIM_ULPClockConfigTypeDef

LPTIM_ULPClockConfigTypeDef is defined in the stm32g4xx_hal_lptim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

35.1.3 LPTIM_TriggerConfigTypeDef

LPTIM_TriggerConfigTypeDef is defined in the stm32g4xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

35.1.4 LPTIM_InitTypeDef

LPTIM_InitTypeDef is defined in the stm32g4xx_hal_lptim.h

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t OutputPolarity***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***
- ***uint32_t Input1Source***
- ***uint32_t Input2Source***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::OutputPolarity***
  Specifies the Output polarity. This parameter can be a value of ***LPTIM_Output_Polarity***
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***
- ***uint32_t LPTIM_InitTypeDef::Input1Source***
  Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of ***LPTIM_Input1_Source***
- ***uint32_t LPTIM_InitTypeDef::Input2Source***
  Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of ***LPTIM_Input2_Source***

35.1.5 LPTIM_HandleTypeDef

LPTIM_HandleTypeDef is defined in the stm32g4xx_hal_lptim.h

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***

Field Documentation

- ***LPTIM_TypeDef* LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef LPTIM_HandleTypeDef::Init***
  LPTIM required parameters
- ***HAL_StatusTypeDef LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status
- ***HAL_LockTypeDef LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef LPTIM_HandleTypeDef::State***
  LPTIM peripheral state

35.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

35.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL_LPTIM_MspInit():
   - Enable the LPTIM interface clock using __HAL_RCC_LPTIMx_CLK_ENABLE().
   - In case of using interrupts (e.g. HAL_LPTIM_PWM_Start_IT()):
     - Configure the LPTIM interrupt priority using HAL_NVIC_SetPriority().
     - Enable the LPTIM IRQ handler using HAL_NVIC_EnableIRQ().
     - In LPTIM IRQ handler, call HAL_LPTIM_IRQHandler().
2. Initialize the LPTIM HAL using HAL_LPTIM_Init(). This function configures mainly:
   - The instance: LPTIM1.
   - Clock: the counter clock.
     - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE or LSI).
     - Prescaler: select the clock divider.
   - UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source.
     - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
     - SampleTime: clock sampling time to configure the clock glitch filter.
   - Trigger: How the counter start.
     - Source: trigger can be software or one of the hardware triggers.
     - ActiveEdge : only for hardware trigger.
     - SampleTime : trigger sampling time to configure the trigger glitch filter.
   - OutputPolarity : 2 opposite polarities are possible.
   - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
   - Input1Source: Source selected for input1 (GPIO or comparator output).
   - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
   - PWM Mode: To generate a PWM signal with specified period and pulse, call HAL_LPTIM_PWM_Start() or HAL_LPTIM_PWM_Start_IT() for interruption mode.
   - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL_LPTIM_OnePulse_Start() or HAL_LPTIM_OnePulse_Start_IT() for interruption mode.
   - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL_LPTIM_SetOnce_Start() or HAL_LPTIM_SetOnce_Start_IT() for interruption mode.
   - Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.
   - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
   - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
5. De-initialize the LPTIM peripheral using HAL_LPTIM_DeInit().

Callback registration

The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_LPTIM_RegisterCallback() to register a callback. @ref HAL_LPTIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_LPTIM_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_LPTIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- MspInitCallback : LPTIM Base Msp Init Callback.
- MspDeInitCallback : LPTIM Base Msp DeInit Callback.
- CompareMatchCallback : Compare match Callback.
- AutoReloadMatchCallback : Auto-reload match Callback.
- TriggerCallback : External trigger event detection Callback.
- CompareWriteCallback : Compare register write complete Callback.
- AutoReloadWriteCallback : Auto-reload register write complete Callback.
- DirectionUpCallback : Up-counting direction change Callback.
- DirectionDownCallback : Down-counting direction change Callback.

By default, after the Init and when the state is HAL_LPTIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples @ref HAL_LPTIM_TriggerCallback(), @ref HAL_LPTIM_CompareMatchCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init/ DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init/DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_LPTIM_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_LPTIM_STATE_READY or HAL_LPTIM_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_LPTIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

35.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize the LPTIM MSP.

This section contains the following APIs:

- ***HAL_LPTIM_Init***
- ***HAL_LPTIM_DeInit***
- ***HAL_LPTIM_MspInit***
- ***HAL_LPTIM_MspDeInit***

35.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.

- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- *HAL_LPTIM_PWM_Start*
- *HAL_LPTIM_PWM_Stop*
- *HAL_LPTIM_PWM_Start_IT*
- *HAL_LPTIM_PWM_Stop_IT*
- *HAL_LPTIM_OnePulse_Start*
- *HAL_LPTIM_OnePulse_Stop*
- *HAL_LPTIM_OnePulse_Start_IT*
- *HAL_LPTIM_OnePulse_Stop_IT*
- *HAL_LPTIM_SetOnce_Start*
- *HAL_LPTIM_SetOnce_Stop*
- *HAL_LPTIM_SetOnce_Start_IT*
- *HAL_LPTIM_SetOnce_Stop_IT*
- *HAL_LPTIM_Encoder_Start*
- *HAL_LPTIM_Encoder_Stop*
- *HAL_LPTIM_Encoder_Start_IT*
- *HAL_LPTIM_Encoder_Stop_IT*
- *HAL_LPTIM_TimeOut_Start*
- *HAL_LPTIM_TimeOut_Stop*
- *HAL_LPTIM_TimeOut_Start_IT*
- *HAL_LPTIM_TimeOut_Stop_IT*
- *HAL_LPTIM_Counter_Start*
- *HAL_LPTIM_Counter_Stop*
- *HAL_LPTIM_Counter_Start_IT*
- *HAL_LPTIM_Counter_Stop_IT*

35.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- *HAL_LPTIM_ReadCounter*
- *HAL_LPTIM_ReadAutoReload*
- *HAL_LPTIM_ReadCompare*

35.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_LPTIM_GetState*

35.2.6 Detailed description of functions

HAL_LPTIM_Init

Function name

HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_DeInit

Function name

HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize the LPTIM peripheral.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_MspInit

Function name

void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_MspDeInit

Function name

void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM in Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM Set once mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Encoder interface.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Counter mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hlptim)

Function description

Return the current counter value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Autoreload (Period) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Autoreload:** value.

HAL_LPTIM_ReadCompare

Function name

uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Compare (Pulse) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Compare:** value.

HAL_LPTIM_IRQHandler

Function name

void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)

Function description

Handle LPTIM interrupt request.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareMatchCallback

Function name

void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name

void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Trigger detected callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Down to Up callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Up to Down callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)

Function description

Return the LPTIM handle state.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** state

LPTIM_Disable

Function name

void LPTIM_Disable (LPTIM_HandleTypeDef * hlptim)

Function description

Disable LPTIM HW instance.

Parameters

- **hlptim:** pointer to a LPTIM_HandleTypeDef structure that contains the configuration information for LPTIM module.

Return values

- **None:**

Notes

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section.

35.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

35.3.1 LPTIM

LPTIM

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

LPTIM Clock Sample Time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

LPTIM Clock Source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

Description:

- Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

Notes:

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section. Please call HAL_LPTIM_GetState() after a call to __HAL_LPTIM_DISABLE to check for TIMEOUT.

__HAL_LPTIM_START_CONTINUOUS

Description:

- Start the LPTIM peripheral in Continuous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

Description:

- Start the LPTIM peripheral in single mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER

Description:

- Reset the LPTIM Counter register in synchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER_AFTERREAD

Description:

- Reset after read of the LPTIM Counter register in asynchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Write the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:

- None

Notes:

- The ARR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_COMPARE_SET

Description:

- Write the passed parameter in the Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Compare value

Return value:

- None

Notes:

- The CMP register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_GET_FLAG

Description:

- Check whether the specified LPTIM flag is set or not.

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be enabled when the LPTIM instance is disabled.

__HAL_LPTIM_DISABLE_IT

Description:

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be disabled when the LPTIM instance is disabled.

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Check whether the specified LPTIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- Interrupt: status.

__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_IT

Description:

- Enable interrupt on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None

__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_IT

Description:

- Disable interrupt on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None

__HAL_LPTIM_WAKEUPTIMER_EXTI_ENABLE_EVENT

Description:

- Enable event on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None.

__HAL_LPTIM_WAKEUPTIMER_EXTI_DISABLE_EVENT

Description:

- Disable event on the LPTIM Wake-up Timer associated Exti line.

Return value:

- None.

LPTIM Exported Types

LPTIM_EXTI_LINE_WAKEUPTIMER_EVENT

External interrupt line 37 Connected to the LPTIM EXTI Line

LPTIM External Trigger Polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

LPTIM Flags Definition

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_CMPOK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM_FLAG_CMPM

LPTIM Input1 Source

LPTIM_INPUT1SOURCE_GPIO

LPTIM_INPUT1SOURCE_COMP1

LPTIM_INPUT1SOURCE_COMP3

LPTIM_INPUT1SOURCE_COMP5

LPTIM_INPUT1SOURCE_COMP7

LPTIM Input2 Source

LPTIM_INPUT2SOURCE_GPIO

LPTIM_INPUT2SOURCE_COMP2

LPTIM_INPUT2SOURCE_COMP4

LPTIM_INPUT2SOURCE_COMP6

LPTIM Interrupts Definition

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_CMPOK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM_IT_CMPM

LPTIM Output Polarity

LPTIM_OUTPUTPOLARITY_HIGH

LPTIM_OUTPUTPOLARITY_LOW

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM_TRIGSOURCE_8

LPTIM_TRIGSOURCE_9

LPTIM_TRIGSOURCE_10

LPTIM_TRIGSOURCE_11

LPTIM_TRIGSOURCE_12

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

36 HAL NAND Generic Driver

36.1 NAND Firmware driver registers structures

36.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32g4xx_hal_nand.h

Data Fields

- ***uint8_t Maker_Id***
- ***uint8_t Device_Id***
- ***uint8_t Third_Id***
- ***uint8_t Fourth_Id***

Field Documentation

- ***uint8_t NAND_IDTypeDef::Maker_Id***
- ***uint8_t NAND_IDTypeDef::Device_Id***
- ***uint8_t NAND_IDTypeDef::Third_Id***
- ***uint8_t NAND_IDTypeDef::Fourth_Id***

36.1.2 NAND_AddressTypeDef

NAND_AddressTypeDef is defined in the stm32g4xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Plane***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Plane***
  NAND memory Zone address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

36.1.3 NAND_DeviceConfigTypeDef

NAND_DeviceConfigTypeDef is defined in the stm32g4xx_hal_nand.h

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t PlaneNbr***
- ***uint32_t PlaneSize***
- ***FunctionalState ExtraCommandEnable***

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
  NAND memory page (without spare area) size measured in bytes for 8 bits adressing or words for 16 bits addressing

- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits adressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages
- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory zone size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
  NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. Example: Toshiba THTH58BYG3S0HBAI6. This parameter could be ENABLE or DISABLE Please check the Read Mode sequnece in the NAND device datasheet

36.1.4 NAND_HandleTypeDef

NAND_HandleTypeDef is defined in the stm32g4xx_hal_nand.h

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***

Field Documentation

- ***FMC_NAND_TypeDef* NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_DeviceConfigTypeDef NAND_HandleTypeDef::Config***
  NAND phusical characteristic information structure

36.2 NAND Firmware driver API description

The following section lists the various functions of the NAND library.

36.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.

- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/ HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/ HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the NAND_DeviceConfigTypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

Note: This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

Callback registration

The compilation define USE_HAL_NAND_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_NAND_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function @ref HAL_NAND_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_NAND_Init and if the state is HAL_NAND_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_NAND_Init and @ref HAL_NAND_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_NAND_Init and @ref HAL_NAND_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_NAND_RegisterCallback before calling @ref HAL_NAND_DeInit or @ref HAL_NAND_Init function. When The compilation define USE_HAL_NAND_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

36.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- ***HAL_NAND_Init***
- ***HAL_NAND_DeInit***
- ***HAL_NAND_MspInit***
- ***HAL_NAND_MspDeInit***
- ***HAL_NAND_IRQHandler***
- ***HAL_NAND_ITCallback***
- ***HAL_NAND_ConfigDevice***
- ***HAL_NAND_Read_ID***

36.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:

- *HAL_NAND_Read_ID*
- *HAL_NAND_Reset*
- *HAL_NAND_ConfigDevice*
- *HAL_NAND_Read_Page_8b*
- *HAL_NAND_Read_Page_16b*
- *HAL_NAND_Write_Page_8b*
- *HAL_NAND_Write_Page_16b*
- *HAL_NAND_Read_SpareArea_8b*
- *HAL_NAND_Read_SpareArea_16b*
- *HAL_NAND_Write_SpareArea_8b*
- *HAL_NAND_Write_SpareArea_16b*
- *HAL_NAND_Erase_Block*
- *HAL_NAND_Address_Inc*

36.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:

- *HAL_NAND_ECC_Enable*
- *HAL_NAND_ECC_Disable*
- *HAL_NAND_GetECC*

36.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

This section contains the following APIs:

- *HAL_NAND_GetState*
- *HAL_NAND_Read_Status*

36.2.6 Detailed description of functions

HAL_NAND_Init

Function name

HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)

Function description

Perform NAND memory Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ComSpace_Timing:** pointer to Common space timing structure
- **AttSpace_Timing:** pointer to Attribute space timing structure

Return values

- **HAL:** status

HAL_NAND_DeInit

Function name

HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)

Function description

Perform NAND memory De-Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ConfigDevice

Function name

HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, NAND_DeviceConfigTypeDef * pDeviceConfig)

Function description

Configure the device: Enter the physical parameters of the device.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pDeviceConfig:** pointer to NAND_DeviceConfigTypeDef structure

Return values

- **HAL:** status

HAL_NAND_Read_ID

Function name

HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)

Function description

Read the NAND memory electronic signature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values

- **HAL:** status

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP Init.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_MspDeInit

Function name

void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP DeInit.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

This function handles NAND device interrupt request.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

NAND interrupt feature callback.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_Reset

Function name

HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)

Function description

NAND memory reset.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_Read_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Read_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Erase_Block

Function name

HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

NAND memory Block erase.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **HAL:** status

HAL_NAND_Address_Inc

Function name

uint32_t HAL_NAND_Address_Inc (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

Increment the NAND memory address.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **The:** new status of the increment address operation. It can be:
  - NAND_VALID_ADDRESS: When the new address is valid address
  - NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_ECC_Enable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)

Function description

Enables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ECC_Disable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)

Function description

Disables dynamically FMC_NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_GetECC

Function name

HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)

Function description

Disables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ECCval:** pointer to ECC value
- **Timeout:** maximum timeout to wait

Return values

- **HAL:** status

HAL_NAND_GetState

Function name

HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnand)

Function description

return the NAND state

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** state

HAL_NAND_Read_Status

Function name

uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)

Function description

NAND memory read status.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **NAND:** status

36.3 NAND Firmware driver defines

The following section lists the various define and macros of the module.

36.3.1 NAND

NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:

- Reset NAND handle state.

Parameters:

- __HANDLE__: specifies the NAND handle.

Return value:

- None

37 HAL NOR Generic Driver

37.1 NOR Firmware driver registers structures

37.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32g4xx_hal_nor.h

Data Fields

- ***uint16_t Manufacturer_Code***
- ***uint16_t Device_Code1***
- ***uint16_t Device_Code2***
- ***uint16_t Device_Code3***

Field Documentation

- ***uint16_t NOR_IDTypeDef::Manufacturer_Code***
  Defines the device's manufacturer code used to identify the memory
- ***uint16_t NOR_IDTypeDef::Device_Code1***
- ***uint16_t NOR_IDTypeDef::Device_Code2***
- ***uint16_t NOR_IDTypeDef::Device_Code3***
  Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set.They can also be accessed by issuing an Auto Select command

37.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32g4xx_hal_nor.h

Data Fields

- ***uint16_t CFI_1***
- ***uint16_t CFI_2***
- ***uint16_t CFI_3***
- ***uint16_t CFI_4***

Field Documentation

- ***uint16_t NOR_CFITypeDef::CFI_1***
  < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***

37.1.3 NOR_HandleTypeDef

NOR_HandleTypeDef is defined in the stm32g4xx_hal_nor.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***

Field Documentation

- ***FMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance***
  Register base address

- ***FMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef NOR_HandleTypeDef::State***
  NOR device access state

37.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

37.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

Note: This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE : NOR memory write data to specified address

Callback registration

The compilation define USE_HAL_NOR_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_NOR_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NOR MspInit.
- MspDeInitCallback : NOR MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function @ref HAL_NOR_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : NOR MspInit.

- MspDeInitCallback : NOR MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_NOR_Init and if the state is HAL_NOR_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_NOR_Init and @ref HAL_NOR_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_NOR_Init and @ref HAL_NOR_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_NOR_RegisterCallback before calling @ref HAL_NOR_DeInit or @ref HAL_NOR_Init function. When The compilation define USE_HAL_NOR_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

37.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Init***
- ***HAL_NOR_DeInit***
- ***HAL_NOR_MspInit***
- ***HAL_NOR_MspDeInit***
- ***HAL_NOR_MspWait***

37.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Read_ID***
- ***HAL_NOR_ReturnToReadMode***
- ***HAL_NOR_Read***
- ***HAL_NOR_Program***
- ***HAL_NOR_ReadBuffer***
- ***HAL_NOR_ProgramBuffer***
- ***HAL_NOR_Erase_Block***
- ***HAL_NOR_Erase_Chip***
- ***HAL_NOR_Read_CFI***

37.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- ***HAL_NOR_WriteOperation_Enable***
- ***HAL_NOR_WriteOperation_Disable***

37.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- ***HAL_NOR_GetState***
- ***HAL_NOR_GetStatus***

37.2.6 Detailed description of functions

HAL_NOR_Init

Function name

HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Perform the NOR memory Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timing:** pointer to NOR control timing structure
- **ExtTiming:** pointer to NOR extended mode timing structure

Return values

- **HAL:** status

HAL_NOR_DeInit

Function name

HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)

Function description

Perform NOR memory De-Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_MspInit

Function name

void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP Init.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspDeInit

Function name

void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP DeInit.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspWait

Function name

void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)

Function description

NOR MSP Wait for Ready/Busy signal.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timeout:** Maximum timeout value

Return values

- **None:**

HAL_NOR_Read_ID

Function name

HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)

Function description

Read NOR flash IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_ID:** pointer to NOR ID structure

Return values

- **HAL:** status

HAL_NOR_ReturnToReadMode

Function name

HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)

Function description

Returns the NOR memory to Read mode.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_Read

Function name

HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Read data from NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** pointer to Device address
- **pData:** pointer to read data

Return values

- **HAL:** status

HAL_NOR_Program

Function name

HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Program data to NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** Device address
- **pData:** pointer to the data to write

Return values

- **HAL:** status

HAL_NOR_ReadBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Reads a half-word buffer from the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** number of Half word to read.

Return values

- **HAL:** status

HAL_NOR_ProgramBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Writes a half-word buffer to the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal start write address
- **pData:** pointer to source data buffer.
- **uwBufferSize:** Size of the buffer to write

Return values

- **HAL:** status

HAL_NOR_Erase_Block

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)

Function description

Erase the specified block of the NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** Block to erase address
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Erase_Chip

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)

Function description

Erase the entire NOR chip.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Read_CFI

Function name

HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)

Function description

Read NOR flash CFI IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_CFI:** pointer to NOR CFI IDs structure

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)

Function description

Enables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)

Function description

Disables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_GetState

Function name

HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)

Function description

return the NOR controller state

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **NOR:** controller state

HAL_NOR_GetStatus

Function name

HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)

Function description

Returns the NOR operation status.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address
- **Timeout:** NOR programming Timeout

Return values

- **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

37.3 NOR Firmware driver defines

The following section lists the various define and macros of the module.

37.3.1 NOR

NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: specifies the NOR handle.

Return value:

- None

38 HAL OPAMP Generic Driver

38.1 OPAMP Firmware driver registers structures

38.1.1 OPAMP_InitTypeDef

OPAMP_InitTypeDef is defined in the stm32g4xx_hal_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t Mode***
- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***FunctionalState InternalOutput***
- ***uint32_t TimerControlledMuxmode***
- ***uint32_t InvertingInputSecondary***
- ***uint32_t NonInvertingInputSecondary***
- ***uint32_t PgaConnect***
- ***uint32_t PgaGain***
- ***uint32_t UserTrimming***
- ***uint32_t TrimmingValueP***
- ***uint32_t TrimmingValueN***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::PowerMode***

  Specifies the power mode Normal or High Speed. This parameter must be a value of ***OPAMP_PowerMode***
- ***uint32_t OPAMP_InitTypeDef::Mode***

  Specifies the OPAMP mode This parameter must be a value of ***OPAMP_Mode*** mode is either Standalone, Follower or PGA
- ***uint32_t OPAMP_InitTypeDef::InvertingInput***

  Specifies the inverting input in Standalone & Pga modes
  - In Standalone mode: i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of ***OPAMP_InvertingInput*** InvertingInput is either VINM0 or VINM1
  - In PGA mode: i.e when mode is OPAMP_PGA_MODE & in Follower mode i.e when mode is OPAMP_FOLLOWER_MODE This parameter is Not Applicable
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInput***

  Specifies the non inverting input of the opamp: This parameter must be a value of ***OPAMP_NonInvertingInput*** NonInvertingInput is either VINP0, VINP1, VINP2 or VINP3
- ***FunctionalState OPAMP_InitTypeDef::InternalOutput***

  Specifies the configuration of the internal output from OPAMP to ADC. This parameter can be ENABLE or DISABLE Note: When this output is enabled, regular output to I/O is disabled
- ***uint32_t OPAMP_InitTypeDef::TimerControlledMuxmode***

  Specifies if the Timer controlled Mux mode is enabled or disabled This parameter must be a single value of ***OPAMP_TimerControlledMuxmode*** or a combination of them to build a more complex switch scheme by using different timers

- ***uint32_t OPAMP_InitTypeDef::InvertingInputSecondary***

  Specifies the inverting input (secondary) of the opamp when TimerControlledMuxmode is enabled i.e. when TimerControlledMuxmode is OPAMP_TIMERCONTROLLEDMUXMODE_ENABLE
  - In Standalone mode: i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of ***OPAMP_InvertingInputSecondary*** InvertingInputSecondary is either VINM0 or VINM1
  - In PGA mode: i.e when mode is OPAMP_PGA_MODE & in Follower mode i.e when mode is OPAMP_FOLLOWER_MODE This parameter must be a value of ***OPAMP_InvertingInputSecondary*** and is used to choose secondary mode (PGA or follower)
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInputSecondary***

  Specifies the non inverting input (secondary) of the opamp when TimerControlledMuxmode is enabled i.e. when TimerControlledMuxmode is OPAMP_TIMERCONTROLLEDMUXMODE_ENABLE This parameter must be a value of ***OPAMP_NonInvertingInputSecondary*** NonInvertingInput is either VINP0, VINP1, VINP2 or VINP3
- ***uint32_t OPAMP_InitTypeDef::PgaConnect***

  Specifies the inverting pin in PGA mode i.e. when mode is OPAMP_PGA_MODE This parameter must be a value of ***OPAMP_PgaConnect*** Either: not connected, connected to VINM0 In this last case, VINM0 can then be used to input signal (negative gain case with or without bias on VINPx) or to input bias (positive gain case with bias)
- ***uint32_t OPAMP_InitTypeDef::PgaGain***

  Specifies the gain in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of ***OPAMP_PgaGain*** (2, 4, 8, 16, 32 or 64) for positive gain & (-1, -3 ,-7, -15, -31 or -63) for negative gain
- ***uint32_t OPAMP_InitTypeDef::UserTrimming***

  Specifies the trimming mode This parameter must be a value of ***OPAMP_UserTrimming*** UserTrimming is either factory or user trimming
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***

  Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***

  Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31

38.1.2 OPAMP_HandleTypeDef

OPAMP_HandleTypeDef is defined in the stm32g4xx_hal_opamp.h

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***

Field Documentation

- ***OPAMP_TypeDef* OPAMP_HandleTypeDef::Instance***

  OPAMP instance's registers base address
- ***OPAMP_InitTypeDef OPAMP_HandleTypeDef::Init***

  OPAMP required parameters
- ***HAL_StatusTypeDef OPAMP_HandleTypeDef::Status***

  OPAMP peripheral status
- ***HAL_LockTypeDef OPAMP_HandleTypeDef::Lock***

  Locking object
- ***__IO HAL_OPAMP_StateTypeDef OPAMP_HandleTypeDef::State***

  OPAMP communication state

38.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

38.2.1 OPAMP Peripheral Features

The device integrates up to 6 operational amplifiers OPAMP1, OPAMP2, OPAMP3, OPAMP4, OPAMP5 and OPAMP6:

1. The OPAMP(s) provides several exclusive running modes.
   - Standalone mode
   - Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - Follower mode
2. The OPAMP(s) provide(s) calibration capabilities.
   - Calibration aims at correcting some offset for running mode.
   - The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
   - HAL_OPAMP_SelfCalibrate:
   - Runs automatically the calibration in 2 steps. (90% of VDDA for NMOS transistors, 10% of VDDA for PMOS transistors). (As OPAMP is Rail-to-rail input/output, these 2 steps calibration is appropriate and enough in most cases).
   - Enables the user trimming mode
   - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
   - for STM32G4 devices having 6 OPAMPs HAL_OPAMPEx_SelfCalibrateAll runs calibration of 6 OPAMPs in parallel.
3. For any running mode, an additional Timer-controlled Mux (multiplexer) mode can be set on top.
   - Timer-controlled Mux mode allows Automatic switching of inputs configuration (inverting and non inverting).
   - Hence on top of defaults (primary) inverting and non-inverting inputs, the user shall select secondary inverting and non inverting inputs.
   - TIM1 OC6, TIM8 OC6 and TIM20 OC6 provides the alternate switching tempo between defaults (primary) and secondary inputs.
   - These 3 timers (TIM1, TIM8 and TIM20) can be combined to design a more complex switching scheme. So that any of the selected channel can initiate the configuration switch.
4. Running mode: Standalone mode
   - Gain is set externally (gain depends on external loads).
   - Follower mode also possible externally by connecting the inverting input to the output.
5. Running mode: Follower mode
   - Inverting Input is not connected.
6. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - The OPAMP(s) output(s) can be internally connected to resistor feedback output.
   - The OPAMP inverting input can be "not" connected, signal to amplify is connected to non inverting input and gain is positive (2,4,8,16,32 or 64)
   - The OPAMP inverting input can be connected to VINM0: If signal is applied to non inverting input, gain is positive (2,4,8,16,32 or 64). If signal is applied to inverting input, gain is negative (-1,-3,-7,-15-,31 or -63). In both cases, the other input can be used as bias.

38.2.2 How to use this driver

High speed / normal power mode

To run in high speed mode:

1. Configure the OPAMP using HAL_OPAMP_Init() function:
   – Select OPAMP_POWERMODE_HIGHSPEED
   – Otherwise select OPAMP_POWERMODE_NORMAL

Calibration

To run the OPAMP calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the OPAMP, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to
   – Configure the OPAMP input AND output in analog mode using HAL_GPIO_Init() to map the OPAMP output to the GPIO pin.
2. Registrate Callbacks
   – The compilation define USE_HAL_OPAMP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.
   – Use Functions @ref HAL_OPAMP_RegisterCallback() to register a user callback, it allows to register following callbacks:
     ◦ MspInitCallback : OPAMP MspInit.
     ◦ MspDeInitCallback : OPAMP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
   – Use function @ref HAL_OPAMP_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
     ◦ MspInitCallback : OPAMP MspInit.
     ◦ MspDeInitCallback : OPAMP MspDeInit.
     ◦ All Callbacks
3. Configure the OPAMP using HAL_OPAMP_Init() function:
   – Select the mode
   – Select the inverting input
   – Select the non-inverting input
   – Select if the internal ouput should be enabled/disabled (if enabled, regular I/O output is disabled)
   – Select if the Timer controlled Mux is disabled or enabled and controlled by specified timer(s)
   – If the Timer controlled Mux mode is enabled, select the secondary inverting input
   – If the Timer controlled Mux mode is enabled, Select the secondary non-inverting input
   – If PGA mode is enabled, Select if inverting input is connected.
   – If PGA mode is enabled, Select PGA gain to be used.
   – Select either factory or user defined trimming mode.
   – If the user defined trimming mode is enabled, select PMOS & NMOS trimming values (typ. settings returned by HAL_OPAMP_SelfCalibrate function).
4. Enable the OPAMP using HAL_OPAMP_Start() function.
5. Disable the OPAMP using HAL_OPAMP_Stop() function.
6. Lock the OPAMP in running mode using HAL_OPAMP_Lock() & HAL_OPAMP_TimerMuxLock functions. From then the configuration can only be modified
   – After HW reset
   – OR thanks to HAL_OPAMP_MspDeInit called (user defined) from HAL_OPAMP_DeInit.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, fill in the HAL_OPAMP_MspInit()
   – This is the case for instance if you wish to use new OPAMP I/O

2. Configure the OPAMP using HAL_OPAMP_Init() function:
   – As in configure case, selects first the parameters you wish to modify.
   – If OPAMP control register is locked, it is not possible to modify any values on the fly (even the timer controlled mux parameters).
   – If OPAMP timer controlled mux mode register is locked, it is possible to modify any values of the control register but none on the timer controlled mux mode one.
3. Change from high speed mode to normal power mode (& vice versa) requires first HAL_OPAMP_DeInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, of OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

38.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_OPAMP_Init***
- ***HAL_OPAMP_DeInit***
- ***HAL_OPAMP_MspInit***
- ***HAL_OPAMP_MspDeInit***

38.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP data transfers.

This section contains the following APIs:

- ***HAL_OPAMP_Start***
- ***HAL_OPAMP_Stop***
- ***HAL_OPAMP_SelfCalibrate***

38.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- ***HAL_OPAMP_Lock***
- ***HAL_OPAMP_LockTimerMux***

38.2.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_OPAMP_GetState***
- ***HAL_OPAMP_GetTrimOffset***

38.2.7 Detailed description of functions

HAL_OPAMP_Init

Function name

HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)

Function description

Initializes the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and initialize the associated handle.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_OPAMP_DeInit

Function name

HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitializes the OPAMP peripheral.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- Deinitialization can't be performed if the OPAMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_OPAMP_MspInit

Function name

void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_MspDeInit

Function name

void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_Start

Function name

HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)

Function description

Start the opamp.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_Stop

Function name

HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)

Function description

Stop the opamp.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)

Function description

Run the self calibration of one OPAMP.

Parameters

- **hopamp:** handle

Return values

- **Updated:** offset trimming values (PMOS & NMOS), user trimming is enabled
- **HAL:** status

Notes

- Calibration is performed in the mode specified in OPAMP init structure (mode normal or high-speed).
- Calibration runs about 25 ms.

HAL_OPAMP_Lock

Function name

HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)

Function description

Lock the selected opamp configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_LockTimerMux

Function name

HAL_StatusTypeDef HAL_OPAMP_LockTimerMux (OPAMP_HandleTypeDef * hopamp)

Function description

Lock the selected opamp timer controlled mux configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_GetState

Function name

HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (OPAMP_HandleTypeDef * hopamp)

Function description

Return the OPAMP state.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** state

HAL_OPAMP_GetTrimOffset

Function name

OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)

Function description

Return the OPAMP factory trimming value.

Parameters

- **hopamp:** OPAMP handle
- **trimmingoffset:** Trimming offset (P or N)

Return values

- **Trimming:** value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available

38.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

38.3.1 OPAMP

OPAMP

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE

Description:

- Reset OPAMP handle state.

Parameters:

- __HANDLE__: OPAMP handle.

Return value:

- None

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY

Dummy trimming value

OPAMP_FACTORYTRIMMING_N

Offset trimming N

OPAMP_FACTORYTRIMMING_P

Offset trimming P

OPAMP Input

OPAMP_INPUT_INVERTING

Inverting input

OPAMP_INPUT_NONINVERTING

Non inverting input

IS_OPAMP_INPUT

OPAMP Inverting Input

OPAMP_INVERTINGINPUT_IO0

Inverting input connected to I/O VINM0 (PA3 for OPAMP1, PA5 for OPAMP2, PB2 for OPAMP3, PB10 for OPAMP4, PB15 for OPAMP5, PA1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_INVERTINGINPUT_IO1

Inverting input connected to I/0 VINM1 (PC5 for OPAMP1, PC5 for OPAMP2, PB10 for OPAMP3, PB8 for OPAMP4, PA3 for OPAMP5, PB1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP Inverting Input Secondary

OPAMP_SEC_INVERTINGINPUT_IO0

OPAMP secondary mode is standalone mode - Only applicable if

OPAMP_SEC_INVERTINGINPUT_IO1

OPAMP secondary mode is standalone mode - Only applicable if

OPAMP_SEC_INVERTINGINPUT_PGA

OPAMP secondary mode is PGA mode - Only applicable if configured mode through call to

OPAMP_SEC_INVERTINGINPUT_FOLLOWER

OPAMP secondary mode is Follower mode - Only applicable if configured mode through call to

OPAMP Mode

OPAMP_STANDALONE_MODE

standalone mode

OPAMP_PGA_MODE

PGA mode

OPAMP_FOLLOWER_MODE

follower mode

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_IO0

Non inverting input connected to I/O VINP0 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4, PB14 for OPAMP5, PB12 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_NONINVERTINGINPUT_IO1

Non inverting input connected to I/O VINP1 (PA3 for OPAMP1, PB14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4, PD12 for OPAMP5, PD9 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_NONINVERTINGINPUT_IO2

Non inverting input connected to I/O VINP2 (PA7 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PB11 for OPAMP4, PC3 for OPAMP5, PB13 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_NONINVERTINGINPUT_IO3

Non inverting input connected to I/O VINP3 (PD14 for OPAMP2)

OPAMP_NONINVERTINGINPUT_DAC

Non inverting input connected internally to DAC channel (DAC3_CH1 for OPAMP1, DAC3_CH2 for OPAMP3, DAC4_CH1 for OPAMP4, DAC4_CH2 for OPAMP5, DAC3_CH1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP Non Inverting Input Secondary

OPAMP_SEC_NONINVERTINGINPUT_IO0

Secondary non inverting input connected to I/O VINP0 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4, PB14 for OPAMP5, PB12 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_SEC_NONINVERTINGINPUT_IO1

Secondary non inverting input connected to I/O VINP1 (PA3 for OPAMP1, PB14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4, PD12 for OPAMP5, PD9 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_SEC_NONINVERTINGINPUT_IO2

Secondary non inverting input connected to I/O VINP2 (PA7 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PB11 for OPAMP4, PC3 for OPAMP5, PB13 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP_SEC_NONINVERTINGINPUT_IO3

Secondary non inverting input connected to I/O VINP3 (PD14 for OPAMP2)

OPAMP_SEC_NONINVERTINGINPUT_DAC

Secondary non inverting input connected internally to DAC channel (DAC3_CH1 for OPAMP1, DAC3_CH2 for OPAMP3, DAC4_CH1 for OPAMP4, DAC4_CH2 for OPAMP5, DAC3_CH1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP Pga Connect

OPAMP_PGA_CONNECT_INVERTINGINPUT_NO

In PGA mode, the inverting input is not connected

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0

In PGA mode, the inverting input is connected to VINM0 for filtering

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0

OPAMP Pga Gain

OPAMP_PGA_GAIN_2_OR_MINUS_1

PGA gain could be 2 or -1

OPAMP_PGA_GAIN_4_OR_MINUS_3

PGA gain could be 4 or -3

OPAMP_PGA_GAIN_8_OR_MINUS_7

PGA gain could be 8 or -7

OPAMP_PGA_GAIN_16_OR_MINUS_15

PGA gain could be 16 or -15

OPAMP_PGA_GAIN_32_OR_MINUS_31

PGA gain could be 32 or -31

OPAMP_PGA_GAIN_64_OR_MINUS_63

PGA gain could be 64 or -63

OPAMP PowerMode

OPAMP_POWERMODE_NORMAL

Output in normal mode

OPAMP_POWERMODE_HIGHSPEED

Output in highspeed mode

OPAMP Timer Controlled Mux mode

OPAMP_TIMERCONTROLLEDMUXMODE_DISABLE

Timer controlled Mux mode disabled

OPAMP_TIMERCONTROLLEDMUXMODE_TIM1_CH6

Timer controlled Mux mode enabled using TIM1 OC6

OPAMP_TIMERCONTROLLEDMUXMODE_TIM8_CH6

Timer controlled Mux mode enabled using TIM8 OC6

OPAMP_TIMERCONTROLLEDMUXMODE_TIM20_CH6

Timer controlled Mux mode enabled using TIM20 OC6 Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY

Factory trimming

OPAMP_TRIMMING_USER

User trimming

OPAMP VREF

OPAMP_VREF_3VDDA

OPAMP Vref = 3.3% VDDA

OPAMP_VREF_10VDDA

OPAMP Vref = 10% VDDA

OPAMP_VREF_50VDDA

OPAMP Vref = 50% VDDA

OPAMP_VREF_90VDDA

OPAMP Vref = 90% VDDA

39 HAL OPAMP Extension Driver

39.1 OPAMPEx Firmware driver API description

The following section lists the various functions of the OPAMPEx library.

39.1.1 Detailed description of functions

HAL_OPAMPEx_SelfCalibrateAll

Function name

HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1, OPAMP_HandleTypeDef * hopamp2, OPAMP_HandleTypeDef * hopamp3, OPAMP_HandleTypeDef * hopamp4, OPAMP_HandleTypeDef * hopamp5, OPAMP_HandleTypeDef * hopamp6)

Function description

Run the self calibration of up to 6 OPAMPs in parallel.

Parameters

- **hopamp1:** handle
- **hopamp2:** handle
- **hopamp3:** handle
- **hopamp4:** handle (1)
- **hopamp5:** handle (1)
- **hopamp6:** handle (1) (1) Parameter not present on STM32GBK1CB/STM32G431xx/STM32G441xx/STM32G471xx devices.

Return values

- **HAL:** status

Notes

- Calibration is performed in the mode specified in OPAMP init structure (mode normal or high-speed).
- Updated offset trimming values (PMOS & NMOS), user trimming is enabled
- Calibration runs about 25 ms.

40 HAL PCD Generic Driver

40.1 PCD Firmware driver registers structures

40.1.1 PCD_HandleTypeDef

PCD_HandleTypeDef is defined in the stm32g4xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Setup***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***
- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t PCD_HandleTypeDef::USB_Address***
  USB Address
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[8]***
  IN endpoint parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::OUT_ep[8]***
  OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
  PCD peripheral status
- ***__IO PCD_StateTypeDef PCD_HandleTypeDef::State***
  PCD communication state
- ***__IO uint32_t PCD_HandleTypeDef::ErrorCode***
  PCD Error code
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
  Setup packet buffer
- ***PCD_LPM_StateTypeDef PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t PCD_HandleTypeDef::BESL***
- ***uint32_t PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE

- ***uint32_t PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler

40.2 PCD Firmware driver API description

The following section lists the various functions of the PCD library.

40.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
   a. Enable the PCD/USB Low Level interface clock using
      ◦ __HAL_RCC_USB_CLK_ENABLE(); For USB Device only FS peripheral
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
   a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
   a. HAL_PCD_Start();

40.2.2 Initialization and de-initialization functions

This section provides functions allowing to:
This section contains the following APIs:

- ***HAL_PCD_Init***
- ***HAL_PCD_DeInit***
- ***HAL_PCD_MspInit***
- ***HAL_PCD_MspDeInit***

40.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.
This section contains the following APIs:

- ***HAL_PCD_Start***
- ***HAL_PCD_Stop***
- ***HAL_PCD_IRQHandler***
- ***HAL_PCD_DataOutStageCallback***
- ***HAL_PCD_DataInStageCallback***
- ***HAL_PCD_SetupStageCallback***
- ***HAL_PCD_SOFCallback***
- ***HAL_PCD_ResetCallback***
- ***HAL_PCD_SuspendCallback***
- ***HAL_PCD_ResumeCallback***
- ***HAL_PCD_ISOOUTIncompleteCallback***
- ***HAL_PCD_ISOINIncompleteCallback***
- ***HAL_PCD_ConnectCallback***
- ***HAL_PCD_DisconnectCallback***

40.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- *HAL_PCD_DevConnect*
- *HAL_PCD_DevDisconnect*
- *HAL_PCD_SetAddress*
- *HAL_PCD_EP_Open*
- *HAL_PCD_EP_Close*
- *HAL_PCD_EP_Receive*
- *HAL_PCD_EP_GetRxCount*
- *HAL_PCD_EP_Transmit*
- *HAL_PCD_EP_SetStall*
- *HAL_PCD_EP_ClrStall*
- *HAL_PCD_EP_Flush*
- *HAL_PCD_ActivateRemoteWakeup*
- *HAL_PCD_DeActivateRemoteWakeup*

40.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_PCD_GetState*

40.2.6 Detailed description of functions

HAL_PCD_Init

Function name

HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and initialize the associated handle.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeInit

Function name

HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes the PCD peripheral.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_MspInit

Function name

void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_MspDeInit

Function name

void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_Start

Function name

HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)

Function description

Start the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Stop

Function name

HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function description

Stop the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_IRQHandler

Function name

void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)

Function description

This function handles PCD interrupt request.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SOFCallback

Function name

void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Start Of Frame callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SetupStageCallback

Function name

void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Setup stage callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResetCallback

Function name

void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Reset callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SuspendCallback

Function name

void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)

Function description

Suspend event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResumeCallback

Function name

void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)

Function description

Resume event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ConnectCallback

Function name

void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Connection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DisconnectCallback

Function name

void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Disconnection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DataOutStageCallback

Function name

void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data OUT stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DataInStageCallback

Function name

void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data IN stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name

void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO OUT callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOINIncompleteCallback

Function name

void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO IN callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DevConnect

Function name

HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)

Function description

Connect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DevDisconnect

Function name

HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)

Function description

Disconnect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetAddress

Function name

HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function description

Set the USB Device address.

Parameters

- **hpcd:** PCD handle
- **address:** new device address

Return values

- **HAL:** status

HAL_PCD_EP_Open

Function name

HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function description

Open and configure an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **ep_mps:** endpoint max packet size
- **ep_type:** endpoint type

Return values

- **HAL:** status

HAL_PCD_EP_Close

Function name

HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Deactivate an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Receive

Function name

HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Receive an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the reception buffer
- **len:** amount of data to be received

Return values

- **HAL:** status

HAL_PCD_EP_Transmit

Function name

HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Send an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the transmission buffer
- **len:** amount of data to be sent

Return values

- **HAL:** status

HAL_PCD_EP_GetRxCount

Function name

uint32_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Get Received Data Size.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **Data:** Size

HAL_PCD_EP_SetStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Set a STALL condition over an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_ClrStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Clear a STALL condition over in an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Flush

Function name

HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Flush an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

Activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

De-activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_GetState

Function name

PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)

Function description

Return the PCD handle state.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** state

40.3 PCD Firmware driver defines

The following section lists the various define and macros of the module.

40.3.1 PCD

PCD

PCD ENDP

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD Endpoint Kind

PCD_SNG_BUF

PCD_DBL_BUF

PCD EP0 MPS

PCD_EP0MPS_64

PCD_EP0MPS_32

PCD_EP0MPS_16

PCD_EP0MPS_08

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

PCD PHY Module

PCD_PHY_ULPI

PCD_PHY_EMBEDDED

PCD_PHY_UTMI

PCD Speed

PCD_SPEED_FULL

41 HAL PCD Extension Driver

41.1 PCDEx Firmware driver API description

The following section lists the various functions of the PCDEx library.

41.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- *HAL_PCDEx_PMAConfig*
- *HAL_PCDEx_ActivateBCD*
- *HAL_PCDEx_DeActivateBCD*
- *HAL_PCDEx_BCD_VBUSDetect*
- *HAL_PCDEx_ActivateLPM*
- *HAL_PCDEx_DeActivateLPM*
- *HAL_PCDEx_LPM_Callback*
- *HAL_PCDEx_BCD_Callback*

41.1.2 Detailed description of functions

HAL_PCDEx_PMAConfig

Function name

HAL_StatusTypeDef HAL_PCDEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaadress)

Function description

Configure PMA for EP.

Parameters

- **hpcd:** Device instance
- **ep_addr:** endpoint address
- **ep_kind:** endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used
- **pmaadress:** EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.

Return values

- **HAL:** status

HAL_PCDEx_ActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Activate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Deactivate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_ActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Activate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Deactivate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_BCD_VBUSDetect

Function name

void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)

Function description

Handle BatteryCharging Process.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_LPM_Callback

Function name

void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)

Function description

Send LPM message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

HAL_PCDEx_BCD_Callback

Function name

void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)

Function description

Send BatteryCharging message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

42 HAL PWR Generic Driver

42.1 PWR Firmware driver registers structures

42.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32g4xx_hal_pwr.h

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***

  PVDLevel: Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_detection_level***.
- ***uint32_t PWR_PVDTypeDef::Mode***

  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***.

42.2 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

42.2.1 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_PWR_DeInit***
- ***HAL_PWR_EnableBkUpAccess***
- ***HAL_PWR_DisableBkUpAccess***

42.2.2 Peripheral Control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in PWR_CR2 register).
- PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through __HAL_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pins are used to wakeup the system from Standby mode or Shutdown mode. The polarity of these pins can be set to configure event detection on high level (rising edge) or low level (falling edge).

Low Power modes configuration

The devices feature 8 low-power modes:

- Low-power Run mode: core and peripherals are running, main regulator off, low power regulator on.
- Sleep mode: Cortex-M4 core stopped, peripherals kept running, main and low power regulators on.
- Low-power Sleep mode: Cortex-M4 core stopped, peripherals kept running, main regulator off, low power regulator on.
- Stop 0 mode: all clocks are stopped except LSI and LSE, main and low power regulators on.
- Stop 1 mode: all clocks are stopped except LSI and LSE, main regulator off, low power regulator on.

- Standby mode with SRAM2: all clocks are stopped except LSI and LSE, SRAM2 content preserved, main regulator off, low power regulator on.
- Standby mode without SRAM2: all clocks are stopped except LSI and LSE, main and low power regulators off.
- Shutdown mode: all clocks are stopped except LSE, main and low power regulators off.

Low-power run mode

- Entry: (from main run mode)
  - set LPR bit with HAL_PWREx_EnableLowPowerRunMode() API after having decreased the system clock below 2 MHz.
- Exit:
  - clear LPR bit then wait for REGLP bit to be reset with HAL_PWREx_DisableLowPowerRunMode() API. Only then can the system clock frequency be increased above 2 MHz.

Sleep mode / Low-power sleep mode

- Entry: The Sleep mode / Low-power Sleep mode is entered thru HAL_PWR_EnterSLEEPMode() API in specifying whether or not the regulator is forced to low-power mode and if exit is interrupt or event-triggered.
  - PWR_MAINREGULATOR_ON: Sleep mode (regulator in main mode).
  - PWR_LOWPOWERREGULATOR_ON: Low-power sleep (regulator in low power mode). In the latter case, the system clock frequency must have been decreased below 2 MHz beforehand.
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- WFI Exit:
  - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) or any wake-up event.
- WFE Exit:
  - Any wake-up event such as an EXTI line configured in event mode.

When exiting the Low-power sleep mode by issuing an interrupt or a wakeup event, the MCU is in Low-power Run mode.

Stop 0, Stop 1 modes

- Entry: The Stop 0, Stop 1 modes are entered thru the following API's:
  - HAL_PWREx_EnterSTOP0Mode() for mode 0 or HAL_PWREx_EnterSTOP1Mode() for mode 1 or for porting reasons HAL_PWR_EnterSTOPMode().
- Regulator setting (applicable to HAL_PWR_EnterSTOPMode() only):
  - PWR_MAINREGULATOR_ON
  - PWR_LOWPOWERREGULATOR_ON
- Exit (interrupt or event-triggered, specified when entering STOP mode):
  - PWR_STOPENTRY_WFI: enter Stop mode with WFI instruction
  - PWR_STOPENTRY_WFE: enter Stop mode with WFE instruction
- WFI Exit:
  - Any EXTI Line (Internal or External) configured in Interrupt mode.
  - Some specific communication peripherals (USART, LPUART, I2C) interrupts when programmed in wakeup mode.
- WFE Exit:
  - Any EXTI Line (Internal or External) configured in Event mode.

When exiting Stop 0 and Stop 1 modes, the MCU is either in Run mode or in Low-power Run mode depending on the LPR bit setting.

Standby mode

The Standby mode offers two options:

- option a) all clocks off except LSI and LSE, RRS bit set (keeps voltage regulator in low power mode). SRAM and registers contents are lost except for the SRAM2 content, the RTC registers, RTC backup registers and Standby circuitry.
- option b) all clocks off except LSI and LSE, RRS bit cleared (voltage regulator then disabled). SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry.
  - Entry:
    - The Standby mode is entered thru HAL_PWR_EnterSTANDBYMode() API. SRAM1 and register contents are lost except for registers in the Backup domain and Standby circuitry. SRAM2 content can be preserved if the bit RRS is set in PWR_CR3 register. To enable this feature, the user can resort to HAL_PWREx_EnableSRAM2ContentRetention() API to set RRS bit.
  - Exit:
    - WKUP pin rising edge, RTC alarm or wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

After waking up from Standby mode, program execution restarts in the same way as after a Reset.

Shutdown mode

In Shutdown mode, voltage regulator is disabled, all clocks are off except LSE, RRS bit is cleared. SRAM and registers contents are lost except for backup domain registers.

- Entry: The Shutdown mode is entered thru HAL_PWREx_EnterSHUTDOWNMode() API.
- Exit:
  - WKUP pin rising edge, RTC alarm or wakeup, tamper event, time-stamp event, external reset in NRST pin.

After waking up from Shutdown mode, program execution restarts in the same way as after a Reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop, Standby and Shutdown modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTCEx_SetTimeStamp_IT() or HAL_RTCEx_SetTamper_IT() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTCEx_SetWakeUpTimer_IT() function.

This section contains the following APIs:

- ***HAL_PWR_ConfigPVD***
- ***HAL_PWR_EnablePVD***
- ***HAL_PWR_DisablePVD***
- ***HAL_PWR_EnableWakeUpPin***
- ***HAL_PWR_DisableWakeUpPin***
- ***HAL_PWR_EnterSLEEPMode***
- ***HAL_PWR_EnterSTOPMode***
- ***HAL_PWR_EnterSTANDBYMode***
- ***HAL_PWR_EnableSleepOnExit***
- ***HAL_PWR_DisableSleepOnExit***
- ***HAL_PWR_EnableSEVOnPend***
- ***HAL_PWR_DisableSEVOnPend***
- ***HAL_PWR_PVDCallback***

42.2.3 Detailed description of functions

HAL_PWR_DeInit

Function name

void HAL_PWR_DeInit (void)

Function description

Deinitialize the HAL PWR peripheral registers to their default reset values.

Return values

- **None:**

HAL_PWR_EnableBkUpAccess

Function name

void HAL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain (RTC registers, RTC backup data registers).

Return values

- **None:**

Notes

- After reset, the backup domain is protected against possible unwanted write accesses.
- RTCSEL that sets the RTC clock source selection is in the RTC back-up domain. In order to set or modify the RTC clock, the backup domain access must be disabled.
- LSEON bit that switches on and off the LSE crystal belongs as well to the back-up domain.

HAL_PWR_DisableBkUpAccess

Function name

void HAL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain (RTC registers, RTC backup data registers).

Return values

- **None:**

HAL_PWR_ConfigPVD

Function name

HAL_StatusTypeDef HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)

Function description

Configure the voltage threshold detected by the Power Voltage Detector (PVD).

Parameters

- **sConfigPVD:** pointer to a PWR_PVDTypeDef structure that contains the PVD configuration information.

Return values

- **None:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage thresholds corresponding to each detection level.

HAL_PWR_EnablePVD

Function name

void HAL_PWR_EnablePVD (void)

Function description

Enable the Power Voltage Detector (PVD).

Return values

- **None:**

HAL_PWR_DisablePVD

Function name

void HAL_PWR_DisablePVD (void)

Function description

Disable the Power Voltage Detector (PVD).

Return values

- **None:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPinPolarity:** Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values which set the default polarity i.e. detection on high level (rising edge):
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5

  or one of the following value where the user can explicitly specify the enabled pin and the chosen polarity:
  - PWR_WAKEUP_PIN1_HIGH or PWR_WAKEUP_PIN1_LOW
  - PWR_WAKEUP_PIN2_HIGH or PWR_WAKEUP_PIN2_LOW
  - PWR_WAKEUP_PIN3_HIGH or PWR_WAKEUP_PIN3_LOW
  - PWR_WAKEUP_PIN4_HIGH or PWR_WAKEUP_PIN4_LOW
  - PWR_WAKEUP_PIN5_HIGH or PWR_WAKEUP_PIN5_LOW

Return values

- **None:**

Notes

- PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.

HAL_PWR_DisableWakeUpPin

Function name

void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5

Return values

- **None:**

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)

Function description

Enter Sleep or Low-power Sleep mode.

Parameters

- **Regulator:** Specifies the regulator state in Sleep/Low-power Sleep mode. This parameter can be one of the following values:
  - PWR_MAINREGULATOR_ON Sleep mode (regulator in main mode)
  - PWR_LOWPOWERREGULATOR_ON Low-power Sleep mode (regulator in low-power mode)
- **SLEEPEntry:** Specifies if Sleep mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_SLEEPENTRY_WFI enter Sleep or Low-power Sleep mode with WFI instruction
  - PWR_SLEEPENTRY_WFE enter Sleep or Low-power Sleep mode with WFE instruction

Return values

- **None:**

Notes

- In Sleep/Low-power Sleep mode, all I/O pins keep the same state as in Run mode.
- Low-power Sleep mode is entered from Low-power Run mode. Therefore, if not yet in Low-power Run mode before calling HAL_PWR_EnterSLEEPMode() with Regulator set to PWR_LOWPOWERREGULATOR_ON, the user can optionally configure the Flash in power-down monde in setting the SLEEP_PD bit in FLASH_ACR register. Additionally, the clock frequency must be reduced below 2 MHz. Setting SLEEP_PD in FLASH_ACR then appropriately reducing the clock frequency must be done before calling HAL_PWR_EnterSLEEPMode() API.
- When exiting Low-power Sleep mode, the MCU is in Low-power Run mode. To move in Run mode, the user must resort to HAL_PWREx_DisableLowPowerRunMode() API.
- When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source.

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)

Function description

Enter Stop mode.

Parameters

- **Regulator:** Specifies the regulator state in Stop mode. This parameter can be one of the following values:
  - PWR_MAINREGULATOR_ON Stop 0 mode (main regulator ON)
  - PWR_LOWPOWERREGULATOR_ON Stop 1 mode (low power regulator ON)
- **STOPEntry:** Specifies Stop 0 or Stop 1 mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI Enter Stop 0 or Stop 1 mode with WFI instruction.
  - PWR_STOPENTRY_WFE Enter Stop 0 or Stop 1 mode with WFE instruction.

Return values

- **None:**

Notes

- This API is named HAL_PWR_EnterSTOPMode to ensure compatibility with legacy code running on devices where only "Stop mode" is mentioned with main or low power regulator ON.
- In Stop mode, all I/O pins keep the same state as in Run mode.
- All clocks in the VCORE domain are stopped; the PLL, the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability (I2Cx, USARTx and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case, the HSI clock is propagated only to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available. The voltage regulator can be configured either in normal (Stop 0) or low-power mode (Stop 1).
- When exiting Stop 0 or Stop 1 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.
- When the voltage regulator operates in low power mode (Stop 1), an additional startup delay is incurred when waking up. By keeping the internal regulator ON during Stop mode (Stop 0), the consumption is higher although the startup time is reduced.

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enter Standby mode.

Return values

- **None:**

Notes

- In Standby mode, the PLL, the HSI and the HSE oscillators are switched off. The voltage regulator is disabled, except when SRAM2 content is preserved in which case the regulator is in low-power mode. SRAM1 and register contents are lost except for registers in the Backup domain and Standby circuitry. SRAM2 content can be preserved if the bit RRS is set in PWR_CR3 register. To enable this feature, the user can resort to HAL_PWREx_EnableSRAM2ContentRetention() API to set RRS bit. The BOR is available.
- The I/Os can be configured either with a pull-up or pull-down or can be kept in analog state. HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() respectively enable Pull Up and Pull Down state, HAL_PWREx_DisableGPIOPullUp() and HAL_PWREx_DisableGPIOPullDown() disable the same. These states are effective in Standby mode only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Indicate Sleep-On-Exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name

void HAL_PWR_DisableSleepOnExit (void)

Function description

Disable Sleep-On-Exit feature when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Clear SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable CORTEX M4 SEVONPEND bit.

Return values

- **None:**

Notes

- Set SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_DisableSEVOnPend

Function name

void HAL_PWR_DisableSEVOnPend (void)

Function description

Disable CORTEX M4 SEVONPEND bit.

Return values

- **None:**

Notes

- Clear SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_PVDCallback

Function name

void HAL_PWR_PVDCallback (void)

Function description

PWR PVD interrupt callback.

Return values

- **None:**

42.3 PWR Firmware driver defines

The following section lists the various define and macros of the module.

42.3.1 PWR

PWR

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check whether or not a specific PWR flag is set.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - PWR_FLAG_WUF1 Wake Up Flag 1. Indicates that a wakeup event was received from the WKUP pin 1.
  - PWR_FLAG_WUF2 Wake Up Flag 2. Indicates that a wakeup event was received from the WKUP pin 2.
  - PWR_FLAG_WUF3 Wake Up Flag 3. Indicates that a wakeup event was received from the WKUP pin 3.
  - PWR_FLAG_WUF4 Wake Up Flag 4. Indicates that a wakeup event was received from the WKUP pin 4.
  - PWR_FLAG_WUF5 Wake Up Flag 5. Indicates that a wakeup event was received from the WKUP pin 5.
  - PWR_FLAG_SB StandBy Flag. Indicates that the system entered StandBy mode.
  - PWR_FLAG_WUFI Wake-Up Flag Internal. Set when a wakeup is detected on the internal wakeup line.
  - PWR_FLAG_REGLPS Low Power Regulator Started. Indicates whether or not the low-power regulator is ready.
  - PWR_FLAG_REGLPF Low Power Regulator Flag. Indicates whether the regulator is ready in main mode or is in low-power mode.
  - PWR_FLAG_VOSF Voltage Scaling Flag. Indicates whether the regulator is ready in the selected voltage range or is still changing to the required voltage level.
  - PWR_FLAG_PVDO Power Voltage Detector Output. Indicates whether VDD voltage is below or above the selected PVD threshold.
  - PWR_FLAG_PVMO3 Peripheral Voltage Monitoring Output 3. Indicates whether VDDA voltage is is below or above PVM3 threshold.
  - PWR_FLAG_PVMO4 Peripheral Voltage Monitoring Output 4. Indicates whether VDDA voltage is is below or above PVM4 threshold.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_PWR_CLEAR_FLAG

Description:

- Clear a specific PWR flag.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - PWR_FLAG_WUF1 Wake Up Flag 1. Indicates that a wakeup event was received from the WKUP pin 1.
  - PWR_FLAG_WUF2 Wake Up Flag 2. Indicates that a wakeup event was received from the WKUP pin 2.
  - PWR_FLAG_WUF3 Wake Up Flag 3. Indicates that a wakeup event was received from the WKUP pin 3.
  - PWR_FLAG_WUF4 Wake Up Flag 4. Indicates that a wakeup event was received from the WKUP pin 4.
  - PWR_FLAG_WUF5 Wake Up Flag 5. Indicates that a wakeup event was received from the WKUP pin 5.
  - PWR_FLAG_WU Encompasses all five Wake Up Flags.
  - PWR_FLAG_SB Standby Flag. Indicates that the system entered Standby mode.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable the PVD Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable the PVD Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:

- Enable the PVD Event Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:

- Disable the PVD Event Line.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVD_EXTI_GET_FLAG

Description:

- Check whether or not the PVD EXTI interrupt flag is set.

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_CLEAR_FLAG

Description:

- Clear the PVD EXTI interrupt flag.

Return value:

- None

Programmable Voltage Detection levels

PWR_PVDLEVEL_0

PVD threshold around 2.0 V

PWR_PVDLEVEL_1

PVD threshold around 2.2 V

PWR_PVDLEVEL_2

PVD threshold around 2.4 V

PWR_PVDLEVEL_3

PVD threshold around 2.5 V

PWR_PVDLEVEL_4

PVD threshold around 2.6 V

PWR_PVDLEVEL_5

PVD threshold around 2.8 V

PWR_PVDLEVEL_6

PVD threshold around 2.9 V

PWR_PVDLEVEL_7

External input analog voltage (compared internally to VREFINT)

PWR PVD event line

PWR_EVENT_LINE_PVD

Event line 16 Connected to the PVD Event Line

PWR PVD external interrupt line

PWR_EXTI_LINE_PVD

External interrupt line 16 Connected to the PVD EXTI Line

PWR PVD interrupt and event mode

PWR_PVD_MODE_NORMAL

Basic mode is used

PWR_PVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_PVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_PVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_MODE_IT

Mask for interruption yielded by PVD threshold crossing

PVD_MODE_EVT

Mask for event yielded by PVD threshold crossing

PVD_RISING_EDGE

Mask for rising edge set as PVD trigger

PVD_FALLING_EDGE

Mask for falling edge set as PVD trigger

PWR regulator mode

PWR_MAINREGULATOR_ON

Regulator in main mode

PWR_LOWPOWERREGULATOR_ON

Regulator in low-power mode

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

Wait For Interruption instruction to enter Sleep mode

PWR_SLEEPENTRY_WFE

Wait For Event instruction to enter Sleep mode

PWR STOP mode entry

PWR_STOPENTRY_WFI

Wait For Interruption instruction to enter Stop mode

PWR_STOPENTRY_WFE

Wait For Event instruction to enter Stop mode

43 HAL PWR Extension Driver

43.1 PWREx Firmware driver registers structures

43.1.1 PWR_PVMTypeDef

PWR_PVMTypeDef is defined in the stm32g4xx_hal_pwr_ex.h

Data Fields

- ***uint32_t PVMType***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVMTypeDef::PVMType***

  PVMType: Specifies which voltage is monitored and against which threshold. This parameter can be a value of ***PWREx_PVM_Type***.
- ***uint32_t PWR_PVMTypeDef::Mode***

  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWREx_PVM_Mode***.

43.2 PWREx Firmware driver API description

The following section lists the various functions of the PWREx library.

43.2.1 Extended Peripheral Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_PWREx_GetVoltageRange***
- ***HAL_PWREx_ControlVoltageScaling***
- ***HAL_PWREx_EnableBatteryCharging***
- ***HAL_PWREx_DisableBatteryCharging***
- ***HAL_PWREx_EnableInternalWakeUpLine***
- ***HAL_PWREx_DisableInternalWakeUpLine***
- ***HAL_PWREx_EnableGPIOPullUp***
- ***HAL_PWREx_DisableGPIOPullUp***
- ***HAL_PWREx_EnableGPIOPullDown***
- ***HAL_PWREx_DisableGPIOPullDown***
- ***HAL_PWREx_EnablePullUpPullDownConfig***
- ***HAL_PWREx_DisablePullUpPullDownConfig***
- ***HAL_PWREx_EnableSRAM2ContentRetention***
- ***HAL_PWREx_DisableSRAM2ContentRetention***
- ***HAL_PWREx_EnablePVM1***
- ***HAL_PWREx_DisablePVM1***
- ***HAL_PWREx_EnablePVM2***
- ***HAL_PWREx_DisablePVM2***
- ***HAL_PWREx_EnablePVM3***
- ***HAL_PWREx_DisablePVM3***
- ***HAL_PWREx_EnablePVM4***
- ***HAL_PWREx_DisablePVM4***
- ***HAL_PWREx_ConfigPVM***
- ***HAL_PWREx_EnableLowPowerRunMode***
- ***HAL_PWREx_DisableLowPowerRunMode***

- *HAL_PWREx_EnterSTOP0Mode*
- *HAL_PWREx_EnterSTOP1Mode*
- *HAL_PWREx_EnterSHUTDOWNMode*
- *HAL_PWREx_PVD_PVM_IRQHandler*
- *HAL_PWREx_PVM1Callback*
- *HAL_PWREx_PVM2Callback*
- *HAL_PWREx_PVM3Callback*
- *HAL_PWREx_PVM4Callback*
- *HAL_PWREx_EnableUCPDStandbyMode*
- *HAL_PWREx_DisableUCPDStandbyMode*
- *HAL_PWREx_EnableUCPDDeadBattery*
- *HAL_PWREx_DisableUCPDDeadBattery*

43.2.2 Detailed description of functions

HAL_PWREx_GetVoltageRange

Function name

uint32_t HAL_PWREx_GetVoltageRange (void)

Function description

Return Voltage Scaling Range.

Return values

- **VOS:** bit field (PWR_REGULATOR_VOLTAGE_SCALE1 or PWR_REGULATOR_VOLTAGE_SCALE2 or PWR_REGULATOR_VOLTAGE_SCALE1_BOOST when applicable)

HAL_PWREx_ControlVoltageScaling

Function name

HAL_StatusTypeDef HAL_PWREx_ControlVoltageScaling (uint32_t VoltageScaling)

Function description

Configure the main internal regulator output voltage.

Parameters

- **VoltageScaling:** specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
  - PWR_REGULATOR_VOLTAGE_SCALE1_BOOST when available, Regulator voltage output range 1 boost mode, typical output voltage at 1.28 V, system frequency up to 170 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE1 Regulator voltage output range 1 mode, typical output voltage at 1.2 V, system frequency up to 150 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2 Regulator voltage output range 2 mode, typical output voltage at 1.0 V, system frequency up to 26 MHz.

Return values

- **HAL:** Status

Notes

- When moving from Range 1 to Range 2, the system frequency must be decreased to a value below 26 MHz before calling HAL_PWREx_ControlVoltageScaling() API. When moving from Range 2 to Range 1, the system frequency can be increased to a value up to 150 MHz after calling HAL_PWREx_ControlVoltageScaling() API. When moving from Range 1 to Boost Mode Range 1, the system frequency can be increased to a value up to 170 MHz after calling HAL_PWREx_ControlVoltageScaling() API.
- When moving from Range 2 to Range 1, the API waits for VOSF flag to be cleared before returning the status. If the flag is not cleared within 50 microseconds, HAL_TIMEOUT status is reported.

HAL_PWREx_EnableBatteryCharging

Function name

void HAL_PWREx_EnableBatteryCharging (uint32_t ResistorSelection)

Function description

Enable battery charging.

Parameters

- **ResistorSelection:** specifies the resistor impedance. This parameter can be one of the following values:
  - PWR_BATTERY_CHARGING_RESISTOR_5 5 kOhms resistor
  - PWR_BATTERY_CHARGING_RESISTOR_1_5 1.5 kOhms resistor

Return values

- **None:**

HAL_PWREx_DisableBatteryCharging

Function name

void HAL_PWREx_DisableBatteryCharging (void)

Function description

Disable battery charging.

Return values

- **None:**

HAL_PWREx_EnableInternalWakeUpLine

Function name

void HAL_PWREx_EnableInternalWakeUpLine (void)

Function description

Enable Internal Wake-up Line.

Return values

- **None:**

HAL_PWREx_DisableInternalWakeUpLine

Function name

void HAL_PWREx_DisableInternalWakeUpLine (void)

Function description

Disable Internal Wake-up Line.

Return values

- **None:**

HAL_PWREx_EnableGPIOPullUp

Function name

HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO:** Specify the IO port. This parameter can be PWR_GPIO_A, ..., PWR_GPIO_G (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to set several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Set the relevant PUy bits of PWR_PUCRx register to configure the I/O in pull-up state in Standby and Shutdown modes.
- This state is effective in Standby and Shutdown modes only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.
- The configuration is lost when exiting the Shutdown mode due to the power-on reset, maintained when exiting the Standby mode.
- To avoid any conflict at Standby and Shutdown modes exits, the corresponding PDy bit of PWR_PDCRx register is cleared unless it is reserved.
- Even if a PUy bit to set is reserved, the other PUy bits entered as input parameter at the same time are set.

HAL_PWREx_DisableGPIOPullUp

Function name

HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-up state in Standby mode and Shutdown modes.

Parameters

- **GPIO:** Specifies the IO port. This parameter can be PWR_GPIO_A, ..., PWR_GPIO_G (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to reset several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Reset the relevant PUy bits of PWR_PUCRx register used to configure the I/O in pull-up state in Standby and Shutdown modes.
- Even if a PUy bit to reset is reserved, the other PUy bits entered as input parameter at the same time are reset.

HAL_PWREx_EnableGPIOPullDown

Function name

HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** Specify the IO port. This parameter can be PWR_GPIO_A..PWR_GPIO_G (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to set several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Set the relevant PDy bits of PWR_PDCRx register to configure the I/O in pull-down state in Standby and Shutdown modes.
- This state is effective in Standby and Shutdown modes only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.
- The configuration is lost when exiting the Shutdown mode due to the power-on reset, maintained when exiting the Standby mode.
- To avoid any conflict at Standby and Shutdown modes exits, the corresponding PUy bit of PWR_PUCRx register is cleared unless it is reserved.
- Even if a PDy bit to set is reserved, the other PDy bits entered as input parameter at the same time are set.

HAL_PWREx_DisableGPIOPullDown

Function name

HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** Specifies the IO port. This parameter can be PWR_GPIO_A..PWR_GPIO_G (or PWR_GPIO_I depending on the devices) to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for the port where less I/O pins are available) or the logical OR of several of them to reset several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Reset the relevant PDy bits of PWR_PDCRx register used to configure the I/O in pull-down state in Standby and Shutdown modes.
- Even if a PDy bit to reset is reserved, the other PDy bits entered as input parameter at the same time are reset.

HAL_PWREx_EnablePullUpPullDownConfig

Function name

void HAL_PWREx_EnablePullUpPullDownConfig (void)

Function description

Enable pull-up and pull-down configuration.

Return values

- **None:**

Notes

- When APC bit is set, the I/O pull-up and pull-down configurations defined in PWR_PUCRx and PWR_PDCRx registers are applied in Standby and Shutdown modes.
- Pull-up set by PUy bit of PWR_PUCRx register is not activated if the corresponding PDy bit of PWR_PDCRx register is also set (pull-down configuration priority is higher). HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() API's ensure there is no conflict when setting PUy or PDy bit.

HAL_PWREx_DisablePullUpPullDownConfig

Function name

void HAL_PWREx_DisablePullUpPullDownConfig (void)

Function description

Disable pull-up and pull-down configuration.

Return values

- **None:**

Notes

- When APC bit is cleared, the I/O pull-up and pull-down configurations defined in PWR_PUCRx and PWR_PDCRx registers are not applied in Standby and Shutdown modes.

HAL_PWREx_EnableSRAM2ContentRetention

Function name

void HAL_PWREx_EnableSRAM2ContentRetention (void)

Function description

Enable SRAM2 content retention in Standby mode.

Return values

- **None:**

Notes

- When RRS bit is set, SRAM2 is powered by the low-power regulator in Standby mode and its content is kept.

HAL_PWREx_DisableSRAM2ContentRetention

Function name

void HAL_PWREx_DisableSRAM2ContentRetention (void)

Function description

Disable SRAM2 content retention in Standby mode.

Return values

- **None:**

Notes

- When RRS bit is reset, SRAM2 is powered off in Standby mode and its content is lost.

HAL_PWREx_EnablePVM1

Function name

void HAL_PWREx_EnablePVM1 (void)

Function description

Enable the Power Voltage Monitoring 1: VDDA versus FASTCOMP minimum voltage.

Return values

- **None:**

HAL_PWREx_DisablePVM1

Function name

void HAL_PWREx_DisablePVM1 (void)

Function description

Disable the Power Voltage Monitoring 1: VDDA versus FASTCOMP minimum voltage.

Return values

- **None:**

HAL_PWREx_EnablePVM2

Function name

void HAL_PWREx_EnablePVM2 (void)

Function description

Enable the Power Voltage Monitoring 2: VDDA versus FASTDAC minimum voltage.

Return values

- **None:**

HAL_PWREx_DisablePVM2

Function name

void HAL_PWREx_DisablePVM2 (void)

Function description

Disable the Power Voltage Monitoring 2: VDDA versus FASTDAC minimum voltage.

Return values

- **None:**

HAL_PWREx_EnablePVM3

Function name

void HAL_PWREx_EnablePVM3 (void)

Function description

Enable the Power Voltage Monitoring 3: VDDA versus ADC minimum voltage 1.62V.

Return values

- **None:**

HAL_PWREx_DisablePVM3

Function name

void HAL_PWREx_DisablePVM3 (void)

Function description

Disable the Power Voltage Monitoring 3: VDDA versus ADC minimum voltage 1.62V.

Return values

- **None:**

HAL_PWREx_EnablePVM4

Function name

void HAL_PWREx_EnablePVM4 (void)

Function description

Enable the Power Voltage Monitoring 4: VDDA versus OPAMP/DAC minimum voltage 1.8V.

Return values

- **None:**

HAL_PWREx_DisablePVM4

Function name

void HAL_PWREx_DisablePVM4 (void)

Function description

Disable the Power Voltage Monitoring 4: VDDA versus OPAMP/DAC minimum voltage 1.8V.

Return values

- **None:**

HAL_PWREx_ConfigPVM

Function name

HAL_StatusTypeDef HAL_PWREx_ConfigPVM (PWR_PVMTypeDef * sConfigPVM)

Function description

Configure the Peripheral Voltage Monitoring (PVM).

Parameters

- **sConfigPVM:** pointer to a PWR_PVMTypeDef structure that contains the PVM configuration information.

Return values

- **HAL:** status

Notes

- The API configures a single PVM according to the information contained in the input structure. To configure several PVMs, the API must be singly called for each PVM used.
- Refer to the electrical characteristics of your device datasheet for more details about the voltage thresholds corresponding to each detection level and to each monitored supply.

HAL_PWREx_EnableLowPowerRunMode

Function name

void HAL_PWREx_EnableLowPowerRunMode (void)

Function description

Enter Low-power Run mode.

Return values

- **None:**

Notes

- In Low-power Run mode, all I/O pins keep the same state as in Run mode.
- When Regulator is set to PWR_LOWPOWERREGULATOR_ON, the user can optionally configure the Flash in power-down monde in setting the RUN_PD bit in FLASH_ACR register. Additionally, the clock frequency must be reduced below 2 MHz. Setting RUN_PD in FLASH_ACR then appropriately reducing the clock frequency must be done before calling HAL_PWREx_EnableLowPowerRunMode() API.

HAL_PWREx_DisableLowPowerRunMode

Function name

HAL_StatusTypeDef HAL_PWREx_DisableLowPowerRunMode (void)

Function description

Exit Low-power Run mode.

Return values

- **HAL:** Status

Notes

- Before HAL_PWREx_DisableLowPowerRunMode() completion, the function checks that REGLPF has been properly reset (otherwise, HAL_PWREx_DisableLowPowerRunMode returns HAL_TIMEOUT status). The system clock frequency can then be increased above 2 MHz.

HAL_PWREx_EnterSTOP0Mode

Function name

void HAL_PWREx_EnterSTOP0Mode (uint8_t STOPEntry)

Function description

Enter Stop 0 mode.

Parameters

- **STOPEntry:** specifies if Stop mode in entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI Enter Stop mode with WFI instruction
  - PWR_STOPENTRY_WFE Enter Stop mode with WFE instruction

Return values

- **None:**

Notes

- In Stop 0 mode, main and low voltage regulators are ON.
- In Stop 0 mode, all I/O pins keep the same state as in Run mode.
- All clocks in the VCORE domain are stopped; the PLL, the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability (I2Cx, USARTx and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case, the HSI clock is propagated only to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available.
- When exiting Stop 0 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock if STOPWUCK bit in RCC_CFGR register is set; the HSI oscillator is selected if STOPWUCK is cleared.
- By keeping the internal regulator ON during Stop 0 mode, the consumption is higher although the startup time is reduced.

HAL_PWREx_EnterSTOP1Mode

Function name

void HAL_PWREx_EnterSTOP1Mode (uint8_t STOPEntry)

Function description

Enter Stop 1 mode.

Parameters

- **STOPEntry:** specifies if Stop mode in entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI Enter Stop mode with WFI instruction
  - PWR_STOPENTRY_WFE Enter Stop mode with WFE instruction

Return values

- **None:**

Notes

- In Stop 1 mode, only low power voltage regulator is ON.
- In Stop 1 mode, all I/O pins keep the same state as in Run mode.
- All clocks in the VCORE domain are stopped; the PLL, the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability (I2Cx, USARTx and LPUART) can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. In this case, the HSI clock is propagated only to the peripheral requesting it. SRAM1, SRAM2 and register contents are preserved. The BOR is available.
- When exiting Stop 1 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock if STOPWUCK bit in RCC_CFGR register is set.
- Due to low power mode, an additional startup delay is incurred when waking up from Stop 1 mode.

HAL_PWREx_EnterSHUTDOWNMode

Function name

void HAL_PWREx_EnterSHUTDOWNMode (void)

Function description

Enter Shutdown mode.

Return values

- **None:**

Notes

- In Shutdown mode, the PLL, the HSI, the LSI and the HSE oscillators are switched off. The voltage regulator is disabled and Vcore domain is powered off. SRAM1, SRAM2 and registers contents are lost except for registers in the Backup domain. The BOR is not available.
- The I/Os can be configured either with a pull-up or pull-down or can be kept in analog state.

HAL_PWREx_PVD_PVM_IRQHandler

Function name

void HAL_PWREx_PVD_PVM_IRQHandler (void)

Function description

This function handles the PWR PVD/PVMx interrupt request.

Return values

- **None:**

Notes

- This API should be called under the PVD_PVM_IRQHandler().

HAL_PWREx_PVM1Callback

Function name

void HAL_PWREx_PVM1Callback (void)

Function description

PWR PVM1 interrupt callback.

Return values

- **None:**

HAL_PWREx_PVM2Callback

Function name

void HAL_PWREx_PVM2Callback (void)

Function description

PWR PVM2 interrupt callback.

Return values

- **None:**

HAL_PWREx_PVM3Callback

Function name

void HAL_PWREx_PVM3Callback (void)

Function description

PWR PVM3 interrupt callback.

Return values

- **None:**

HAL_PWREx_PVM4Callback

Function name

void HAL_PWREx_PVM4Callback (void)

Function description

PWR PVM4 interrupt callback.

Return values

- **None:**

HAL_PWREx_EnableUCPDStandbyMode

Function name

void HAL_PWREx_EnableUCPDStandbyMode (void)

Function description

Enable UCPD configuration memorization in Standby.

Return values

- **None:**

HAL_PWREx_DisableUCPDStandbyMode

Function name

void HAL_PWREx_DisableUCPDStandbyMode (void)

Function description

Disable UCPD configuration memorization in Standby.

Return values

- **None:**

Notes

- This function must be called on exiting the Standby mode and before any UCPD configuration update.

HAL_PWREx_EnableUCPDDeadBattery

Function name

void HAL_PWREx_EnableUCPDDeadBattery (void)

Function description

Enable the USB Type-C dead battery pull-down behavior on UCPDx_CC1 and UCPDx_CC2 pins.

Return values

- **None:**

HAL_PWREx_DisableUCPDDeadBattery

Function name

void HAL_PWREx_DisableUCPDDeadBattery (void)

Function description

Disable the USB Type-C dead battery pull-down behavior on UCPDx_CC1 and UCPDx_CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior will be enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

43.3 PWREx Firmware driver defines

The following section lists the various define and macros of the module.

43.3.1 PWREx

PWREx

PWR Extended Exported Macros

__HAL_PWR_PVM1_EXTI_ENABLE_IT

Description:

- Enable the PVM1 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_IT

Description:

- Disable the PVM1 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_EVENT

Description:

- Enable the PVM1 Event Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_EVENT

Description:

- Disable the PVM1 Event Line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVM1 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM1 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM1 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM1 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM1 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVM1 Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVM1_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVM1_EXTI_GET_FLAG

Description:

- Check whether the specified PVM1 EXTI interrupt flag is set or not.

Return value:

- EXTI: PVM1 Line Status.

__HAL_PWR_PVM1_EXTI_CLEAR_FLAG

Description:

- Clear the PVM1 EXTI flag.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_IT

Description:

- Enable the PVM2 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_IT

Description:

- Disable the PVM2 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_EVENT

Description:

- Enable the PVM2 Event Line.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_EVENT

Description:

- Disable the PVM2 Event Line.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVM2 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM2 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM2 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM2 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM2 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVM2 Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVM2_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVM2_EXTI_GET_FLAG

Description:

- Check whether the specified PVM2 EXTI interrupt flag is set or not.

Return value:

- EXTI: PVM2 Line Status.

__HAL_PWR_PVM2_EXTI_CLEAR_FLAG

Description:

- Clear the PVM2 EXTI flag.

Return value:

- None

__HAL_PWR_PVM3_EXTI_ENABLE_IT

Description:

- Enable the PVM3 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM3_EXTI_DISABLE_IT

Description:

- Disable the PVM3 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM3_EXTI_ENABLE_EVENT

Description:

- Enable the PVM3 Event Line.

Return value:

- None

__HAL_PWR_PVM3_EXTI_DISABLE_EVENT

Description:

- Disable the PVM3 Event Line.

Return value:

- None

__HAL_PWR_PVM3_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVM3 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM3_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM3 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM3_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM3 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM3_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM3 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM3_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM3 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM3_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVM3 Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVM3_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVM3_EXTI_GET_FLAG

Description:

- Check whether the specified PVM3 EXTI interrupt flag is set or not.

Return value:

- EXTI: PVM3 Line Status.

__HAL_PWR_PVM3_EXTI_CLEAR_FLAG

Description:

- Clear the PVM3 EXTI flag.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_IT

Description:

- Enable the PVM4 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_IT

Description:

- Disable the PVM4 Extended Interrupt Line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_EVENT

Description:

- Enable the PVM4 Event Line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_EVENT

Description:

- Disable the PVM4 Event Line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVM4 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVM4 Extended Interrupt Rising Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVM4 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVM4 Extended Interrupt Falling Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- PVM4 EXTI line configuration: set rising & falling edge trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVM4 Extended Interrupt Rising & Falling Trigger.

Return value:

- None

__HAL_PWR_PVM4_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVM4_EXTI_GET_FLAG

Description:

- Check whether or not the specified PVM4 EXTI interrupt flag is set.

Return value:

- EXTI: PVM4 Line Status.

__HAL_PWR_PVM4_EXTI_CLEAR_FLAG

Description:

- Clear the PVM4 EXTI flag.

Return value:

- None

__HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- Configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
  - PWR_REGULATOR_VOLTAGE_SCALE1_BOOST Regulator voltage output range 1 mode, typical output voltage at 1.28 V, system frequency up to 170 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE1 Regulator voltage output range 1 mode, typical output voltage at 1.2 V, system frequency up to 150 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2 Regulator voltage output range 2 mode, typical output voltage at 1.0 V, system frequency up to 26 MHz.

Return value:

- None

Notes:

- This macro is similar to HAL_PWREx_ControlVoltageScaling() API but doesn't check whether or not VOSF flag is cleared when moving from range 2 to range 1. User may resort to __HAL_PWR_GET_FLAG() macro to check VOSF bit resetting.

PWR Status Flags

PWR_FLAG_WUF1

Wakeup event on wakeup pin 1

PWR_FLAG_WUF2

Wakeup event on wakeup pin 2

PWR_FLAG_WUF3

Wakeup event on wakeup pin 3

PWR_FLAG_WUF4

Wakeup event on wakeup pin 4

PWR_FLAG_WUF5

Wakeup event on wakeup pin 5

PWR_FLAG_WU

Encompass wakeup event on all wakeup pins

PWR_FLAG_SB

Standby flag

PWR_FLAG_WUFI

Wakeup on internal wakeup line

PWR_FLAG_REGLPS

Low-power regulator start flag

PWR_FLAG_REGLPF

Low-power regulator flag

PWR_FLAG_VOSF

Voltage scaling flag

PWR_FLAG_PVDO

Power Voltage Detector output flag

PWR_FLAG_PVMO1

Power Voltage Monitoring 1 output flag

PWR_FLAG_PVMO2

Power Voltage Monitoring 2 output flag

PWR_FLAG_PVMO3

Power Voltage Monitoring 3 output flag

PWR_FLAG_PVMO4

Power Voltage Monitoring 4 output flag

GPIO port

PWR_GPIO_A

GPIO port A

PWR_GPIO_B

GPIO port B

PWR_GPIO_C

GPIO port C

PWR_GPIO_D

GPIO port D

PWR_GPIO_E

GPIO port E

PWR_GPIO_F

GPIO port F

PWR_GPIO_G

GPIO port G

GPIO bit number for I/O setting in standby/shutdown mode

PWR_GPIO_BIT_0

GPIO port I/O pin 0

PWR_GPIO_BIT_1

GPIO port I/O pin 1

PWR_GPIO_BIT_2

GPIO port I/O pin 2

PWR_GPIO_BIT_3

GPIO port I/O pin 3

PWR_GPIO_BIT_4

GPIO port I/O pin 4

PWR_GPIO_BIT_5

GPIO port I/O pin 5

PWR_GPIO_BIT_6

GPIO port I/O pin 6

PWR_GPIO_BIT_7

GPIO port I/O pin 7

PWR_GPIO_BIT_8

GPIO port I/O pin 8

PWR_GPIO_BIT_9

GPIO port I/O pin 9

PWR_GPIO_BIT_10

GPIO port I/O pin 10

PWR_GPIO_BIT_11

GPIO port I/O pin 11

PWR_GPIO_BIT_12

GPIO port I/O pin 12

PWR_GPIO_BIT_13

GPIO port I/O pin 13

PWR_GPIO_BIT_14

GPIO port I/O pin 14

PWR_GPIO_BIT_15

GPIO port I/O pin 15

PWR PVM event lines

PWR_EVENT_LINE_PVM1

Event line 35 Connected to the PVM1 EXTI Line

PWR_EVENT_LINE_PVM2

Event line 36 Connected to the PVM2 EXTI Line

PWR_EVENT_LINE_PVM3

Event line 37 Connected to the PVM3 EXTI Line

PWR_EVENT_LINE_PVM4

Event line 38 Connected to the PVM4 EXTI Line

PWR PVM external interrupts lines

PWR_EXTI_LINE_PVM1

External interrupt line 35 Connected to the PVM1 EXTI Line

PWR_EXTI_LINE_PVM2

External interrupt line 36 Connected to the PVM2 EXTI Line

PWR_EXTI_LINE_PVM3

External interrupt line 37 Connected to the PVM3 EXTI Line

PWR_EXTI_LINE_PVM4

External interrupt line 38 Connected to the PVM4 EXTI Line

PWR PVM interrupt and event mode

PWR_PVM_MODE_NORMAL

basic mode is used

PWR_PVM_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_PVM_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_PVM_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVM_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVM_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVM_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVM Mode Mask

PVM_MODE_IT

Mask for interruption yielded by PVM threshold crossing

PVM_MODE_EVT

Mask for event yielded by PVM threshold crossing

PVM_RISING_EDGE

Mask for rising edge set as PVM trigger

PVM_FALLING_EDGE

Mask for falling edge set as PVM trigger

Peripheral Voltage Monitoring type

PWR_PVM_1

Peripheral Voltage Monitoring 1 enable: VDDUSB versus 1.2 V (applicable when USB feature is supported)

PWR_PVM_2

Peripheral Voltage Monitoring 2 enable: VDDIO2 versus 0.9 V (applicable when VDDIO2 is present on device)

PWR_PVM_3

Peripheral Voltage Monitoring 3 enable: VDDA versus 1.62 V

PWR_PVM_4

Peripheral Voltage Monitoring 4 enable: VDDA versus 2.2 V

PWR Regulator voltage scale

PWR_REGULATOR_VOLTAGE_SCALE1_BOOST

Voltage scaling range 1 boost mode

PWR_REGULATOR_VOLTAGE_SCALE1

Voltage scaling range 1 normal mode

PWR_REGULATOR_VOLTAGE_SCALE2

Voltage scaling range 2

PWR Extended Flag Setting Time Out Value

PWR_FLAG_SETTING_DELAY_US

Time out value for REGLPF and VOSF flags setting

PWR battery charging

PWR_BATTERY_CHARGING_DISABLE

PWR_BATTERY_CHARGING_ENABLE

PWR battery charging resistor selection

PWR_BATTERY_CHARGING_RESISTOR_5

VBAT charging through a 5 kOhms resistor

PWR_BATTERY_CHARGING_RESISTOR_1_5

VBAT charging through a 1.5 kOhms resistor

PWR wake-up pins

PWR_WAKEUP_PIN1

Wakeup pin 1 (with high level polarity)

PWR_WAKEUP_PIN2

Wakeup pin 2 (with high level polarity)

PWR_WAKEUP_PIN3

Wakeup pin 3 (with high level polarity)

PWR_WAKEUP_PIN4

Wakeup pin 4 (with high level polarity)

PWR_WAKEUP_PIN5

Wakeup pin 5 (with high level polarity)

PWR_WAKEUP_PIN1_HIGH

Wakeup pin 1 (with high level polarity)

PWR_WAKEUP_PIN2_HIGH

Wakeup pin 2 (with high level polarity)

PWR_WAKEUP_PIN3_HIGH

Wakeup pin 3 (with high level polarity)

PWR_WAKEUP_PIN4_HIGH

Wakeup pin 4 (with high level polarity)

PWR_WAKEUP_PIN5_HIGH

Wakeup pin 5 (with high level polarity)

PWR_WAKEUP_PIN1_LOW

Wakeup pin 1 (with low level polarity)

PWR_WAKEUP_PIN2_LOW

Wakeup pin 2 (with low level polarity)

PWR_WAKEUP_PIN3_LOW

Wakeup pin 3 (with low level polarity)

PWR_WAKEUP_PIN4_LOW

Wakeup pin 4 (with low level polarity)

PWR_WAKEUP_PIN5_LOW

Wakeup pin 5 (with low level polarity)

Shift to apply to retrieve polarity information from PWR_WAKEUP_PINy_xxx constants

PWR_WUP_POLARITY_SHIFT

Internal constant used to retrieve wakeup pin polariry

44 HAL QSPI Generic Driver

44.1 QSPI Firmware driver registers structures

44.1.1 QSPI_InitTypeDef

QSPI_InitTypeDef is defined in the stm32g4xx_hal_qspi.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t FifoThreshold***
- ***uint32_t SampleShifting***
- ***uint32_t FlashSize***
- ***uint32_t ChipSelectHighTime***
- ***uint32_t ClockMode***
- ***uint32_t FlashID***
- ***uint32_t DualFlash***

Field Documentation

- ***uint32_t QSPI_InitTypeDef::ClockPrescaler***
- ***uint32_t QSPI_InitTypeDef::FifoThreshold***
- ***uint32_t QSPI_InitTypeDef::SampleShifting***
- ***uint32_t QSPI_InitTypeDef::FlashSize***
- ***uint32_t QSPI_InitTypeDef::ChipSelectHighTime***
- ***uint32_t QSPI_InitTypeDef::ClockMode***
- ***uint32_t QSPI_InitTypeDef::FlashID***
- ***uint32_t QSPI_InitTypeDef::DualFlash***

44.1.2 QSPI_HandleTypeDef

QSPI_HandleTypeDef is defined in the stm32g4xx_hal_qspi.h

Data Fields

- ***QUADSPI_TypeDef * Instance***
- ***QSPI_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***__IO uint32_t TxXferSize***
- ***__IO uint32_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***__IO uint32_t RxXferSize***
- ***__IO uint32_t RxXferCount***
- ***DMA_HandleTypeDef * hdma***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_QSPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***

Field Documentation

- ***QUADSPI_TypeDef* QSPI_HandleTypeDef::Instance***
- ***QSPI_InitTypeDef QSPI_HandleTypeDef::Init***
- ***uint8_t* QSPI_HandleTypeDef::pTxBuffPtr***

- *__IO uint32_t QSPI_HandleTypeDef::TxXferSize*
- *__IO uint32_t QSPI_HandleTypeDef::TxXferCount*
- *uint8_t* QSPI_HandleTypeDef::pRxBuffPtr*
- *__IO uint32_t QSPI_HandleTypeDef::RxXferSize*
- *__IO uint32_t QSPI_HandleTypeDef::RxXferCount*
- *DMA_HandleTypeDef* QSPI_HandleTypeDef::hdma*
- *__IO HAL_LockTypeDef QSPI_HandleTypeDef::Lock*
- *__IO HAL_QSPI_StateTypeDef QSPI_HandleTypeDef::State*
- *__IO uint32_t QSPI_HandleTypeDef::ErrorCode*
- *uint32_t QSPI_HandleTypeDef::Timeout*

44.1.3 QSPI_CommandTypeDef

QSPI_CommandTypeDef is defined in the stm32g4xx_hal_qspi.h

Data Fields

- *uint32_t Instruction*
- *uint32_t Address*
- *uint32_t AlternateBytes*
- *uint32_t AddressSize*
- *uint32_t AlternateBytesSize*
- *uint32_t DummyCycles*
- *uint32_t InstructionMode*
- *uint32_t AddressMode*
- *uint32_t AlternateByteMode*
- *uint32_t DataMode*
- *uint32_t NbData*
- *uint32_t DdrMode*
- *uint32_t DdrHoldHalfCycle*
- *uint32_t SIOOMode*

Field Documentation

- *uint32_t QSPI_CommandTypeDef::Instruction*
- *uint32_t QSPI_CommandTypeDef::Address*
- *uint32_t QSPI_CommandTypeDef::AlternateBytes*
- *uint32_t QSPI_CommandTypeDef::AddressSize*
- *uint32_t QSPI_CommandTypeDef::AlternateBytesSize*
- *uint32_t QSPI_CommandTypeDef::DummyCycles*
- *uint32_t QSPI_CommandTypeDef::InstructionMode*
- *uint32_t QSPI_CommandTypeDef::AddressMode*
- *uint32_t QSPI_CommandTypeDef::AlternateByteMode*
- *uint32_t QSPI_CommandTypeDef::DataMode*
- *uint32_t QSPI_CommandTypeDef::NbData*
- *uint32_t QSPI_CommandTypeDef::DdrMode*
- *uint32_t QSPI_CommandTypeDef::DdrHoldHalfCycle*
- *uint32_t QSPI_CommandTypeDef::SIOOMode*

44.1.4 QSPI_AutoPollingTypeDef

QSPI_AutoPollingTypeDef is defined in the stm32g4xx_hal_qspi.h

Data Fields

- ***uint32_t Match***
- ***uint32_t Mask***
- ***uint32_t Interval***
- ***uint32_t StatusBytesSize***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***

Field Documentation

- ***uint32_t QSPI_AutoPollingTypeDef::Match***
- ***uint32_t QSPI_AutoPollingTypeDef::Mask***
- ***uint32_t QSPI_AutoPollingTypeDef::Interval***
- ***uint32_t QSPI_AutoPollingTypeDef::StatusBytesSize***
- ***uint32_t QSPI_AutoPollingTypeDef::MatchMode***
- ***uint32_t QSPI_AutoPollingTypeDef::AutomaticStop***

44.1.5 QSPI_MemoryMappedTypeDef

QSPI_MemoryMappedTypeDef is defined in the stm32g4xx_hal_qspi.h

Data Fields

- ***uint32_t TimeOutPeriod***
- ***uint32_t TimeOutActivation***

Field Documentation

- ***uint32_t QSPI_MemoryMappedTypeDef::TimeOutPeriod***
- ***uint32_t QSPI_MemoryMappedTypeDef::TimeOutActivation***

44.2 QSPI Firmware driver API description

The following section lists the various functions of the QSPI library.

44.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_QSPI_MspInit() :
   - Enable QuadSPI clock interface with __HAL_RCC_QSPI_CLK_ENABLE().
   - Reset QuadSPI Peripheral with __HAL_RCC_QSPI_FORCE_RESET() and __HAL_RCC_QSPI_RELEASE_RESET().
   - Enable the clocks for the QuadSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   - Configure these QuadSPI pins in alternate mode using HAL_GPIO_Init().
   - If interrupt mode is used, enable and configure QuadSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   - If DMA mode is used, enable the clocks for the QuadSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with QuadSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the flash size, the clock prescaler, the fifo threshold, the clock mode, the sample shifting and the CS high time using the HAL_QSPI_Init() function.

Indirect functional mode

1. Configure the command sequence using the HAL_QSPI_Command() or HAL_QSPI_Command_IT() functions :
   - Instruction phase : the mode used and if present the instruction opcode.
   - Address phase : the mode used and if present the size and the address value.
   - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
   - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
   - Data phase : the mode used and if present the number of bytes.
   - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
   - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
2. If no data is required for the command, it is sent directly to the memory :
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_CmdCpltCallback() will be called when the transfer is complete.
3. For the indirect write mode, use HAL_QSPI_Transmit(), HAL_QSPI_Transmit_DMA() or HAL_QSPI_Transmit_IT() after the command configuration :
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
   - In DMA mode, HAL_QSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
4. For the indirect read mode, use HAL_QSPI_Receive(), HAL_QSPI_Receive_DMA() or HAL_QSPI_Receive_IT() after the command configuration :
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.
   - In DMA mode, HAL_QSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

1. Configure the command sequence and the auto-polling functional mode using the HAL_QSPI_AutoPolling() or HAL_QSPI_AutoPolling_IT() functions :
   - Instruction phase : the mode used and if present the instruction opcode.
   - Address phase : the mode used and if present the size and the address value.
   - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
   - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
   - Data phase : the mode used.
   - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
   - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
   - The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.
2. After the configuration :
   - In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
   - In interrupt mode, HAL_QSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

1. Configure the command sequence and the memory-mapped functional mode using the HAL_QSPI_MemoryMapped() functions :
   – Instruction phase : the mode used and if present the instruction opcode.
   – Address phase : the mode used and the size.
   – Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
   – Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
   – Data phase : the mode used.
   – Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
   – Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
   – The timeout activation and the timeout period.
2. After the configuration, the QuadSPI will be used as soon as an access on the AHB is done on the address range. HAL_QSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

1. HAL_QSPI_GetError() function gives the error raised during the last operation.
2. HAL_QSPI_Abort() and HAL_QSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :
   – In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
   – In interrupt mode, HAL_QSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

1. HAL_QSPI_GetState() function gives the current state of the HAL QuadSPI driver.
2. HAL_QSPI_SetTimeout() function configures the timeout value used in the driver.
3. HAL_QSPI_SetFifoThreshold() function configures the threshold on the Fifo of the QSPI IP.
4. HAL_QSPI_GetFifoThreshold() function gives the current of the Fifo's threshold
5. HAL_QSPI_SetFlashID() function configures the index of the flash memory to be accessed.

Callback registration

The compilation define USE_HAL_QSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_QSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : QSPI MspInit.
- MspDeInitCallback : QSPI MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function @ref HAL_QSPI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.

- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : QSPI MspInit.
- MspDeInitCallback : QSPI MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_QSPI_Init and if the state is HAL_QSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_QSPI_Init and @ref HAL_QSPI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_QSPI_Init and @ref HAL_QSPI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_QSPI_RegisterCallback before calling @ref HAL_QSPI_DeInit or @ref HAL_QSPI_Init function. When The compilation define USE_HAL_QSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Workarounds linked to Silicon Limitation

1. Workarounds Implemented inside HAL Driver
   – Extra data written in the FIFO at the end of a read transfer

44.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the QuadSPI.
- De-initialize the QuadSPI.

This section contains the following APIs:

- ***HAL_QSPI_Init***
- ***HAL_QSPI_DeInit***
- ***HAL_QSPI_MspInit***
- ***HAL_QSPI_MspDeInit***

44.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- ***HAL_QSPI_IRQHandler***
- ***HAL_QSPI_Command***
- ***HAL_QSPI_Command_IT***
- ***HAL_QSPI_Transmit***
- ***HAL_QSPI_Receive***
- ***HAL_QSPI_Transmit_IT***
- ***HAL_QSPI_Receive_IT***

- *HAL_QSPI_Transmit_DMA*
- *HAL_QSPI_Receive_DMA*
- *HAL_QSPI_AutoPolling*
- *HAL_QSPI_AutoPolling_IT*
- *HAL_QSPI_MemoryMapped*
- *HAL_QSPI_ErrorCallback*
- *HAL_QSPI_AbortCpltCallback*
- *HAL_QSPI_CmdCpltCallback*
- *HAL_QSPI_RxCpltCallback*
- *HAL_QSPI_TxCpltCallback*
- *HAL_QSPI_RxHalfCpltCallback*
- *HAL_QSPI_TxHalfCpltCallback*
- *HAL_QSPI_FifoThresholdCallback*
- *HAL_QSPI_StatusMatchCallback*
- *HAL_QSPI_TimeOutCallback*

44.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.

This section contains the following APIs:

- *HAL_QSPI_GetState*
- *HAL_QSPI_GetError*
- *HAL_QSPI_Abort*
- *HAL_QSPI_Abort_IT*
- *HAL_QSPI_SetTimeout*
- *HAL_QSPI_SetFifoThreshold*
- *HAL_QSPI_GetFifoThreshold*
- *HAL_QSPI_SetFlashID*

44.2.5 Detailed description of functions

HAL_QSPI_Init

Function name

HAL_StatusTypeDef HAL_QSPI_Init (QSPI_HandleTypeDef * hqspi)

Function description

Initialize the QSPI mode according to the specified parameters in the QSPI_InitTypeDef and initialize the associated handle.

Parameters

- **hqspi:** : QSPI handle

Return values

- **HAL:** status

HAL_QSPI_DeInit

Function name

HAL_StatusTypeDef HAL_QSPI_DeInit (QSPI_HandleTypeDef * hqspi)

Function description

De-Initialize the QSPI peripheral.

Parameters

- **hqspi:** : QSPI handle

Return values

- **HAL:** status

HAL_QSPI_MspInit

Function name

void HAL_QSPI_MspInit (QSPI_HandleTypeDef * hqspi)

Function description

Initialize the QSPI MSP.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_MspDeInit

Function name

void HAL_QSPI_MspDeInit (QSPI_HandleTypeDef * hqspi)

Function description

DeInitialize the QSPI MSP.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_IRQHandler

Function name

void HAL_QSPI_IRQHandler (QSPI_HandleTypeDef * hqspi)

Function description

Handle QSPI interrupt request.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_Command

Function name

HAL_StatusTypeDef HAL_QSPI_Command (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hqspi:** : QSPI handle
- **cmd:** : structure that contains the command configuration information
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hqspi:** : QSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive

Function name

HAL_StatusTypeDef HAL_QSPI_Receive (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hqspi:** : QSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Command_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd)

Function description

Set the command configuration in interrupt mode.

Parameters

- **hqspi:** : QSPI handle
- **cmd:** : structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hqspi:** : QSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Receive_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hqspi:** : QSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hqspi:** : QSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_QSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_QSPI_Receive_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hqspi:** : QSPI handle
- **pData:** : pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_QSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_QSPI_AutoPolling (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)

Function description

Configure the QSPI Automatic Polling Mode in blocking mode.

Parameters

- **hqspi:** : QSPI handle
- **cmd:** : structure that contains the command configuration information.
- **cfg:** : structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_QSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_QSPI_AutoPolling_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg)

Function description

Configure the QSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hqspi:** : QSPI handle
- **cmd:** : structure that contains the command configuration information.
- **cfg:** : structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_QSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_QSPI_MemoryMapped (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_MemoryMappedTypeDef * cfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hqspi:** : QSPI handle
- **cmd:** : structure that contains the command configuration information.
- **cfg:** : structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_QSPI_ErrorCallback

Function name

void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hqspi)

Function description

Transfer Error callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_AbortCpltCallback

Function name

void HAL_QSPI_AbortCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Abort completed callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_FifoThresholdCallback

Function name

void HAL_QSPI_FifoThresholdCallback (QSPI_HandleTypeDef * hqspi)

Function description

FIFO Threshold callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_CmdCpltCallback

Function name

void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Command completed callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_RxCpltCallback

Function name

void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Rx Transfer completed callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_TxCpltCallback

Function name

void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Tx Transfer completed callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_RxHalfCpltCallback

Function name

void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_TxHalfCpltCallback

Function name

void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_StatusMatchCallback

Function name

void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDef * hqspi)

Function description

Status Match callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_TimeOutCallback

Function name

void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDef * hqspi)

Function description

Timeout callback.

Parameters

- **hqspi:** : QSPI handle

Return values

- **None:**

HAL_QSPI_GetState

Function name

HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDef * hqspi)

Function description

Return the QSPI handle state.

Parameters

- **hqspi:** : QSPI handle

Return values

- **HAL:** state

HAL_QSPI_GetError

Function name

uint32_t HAL_QSPI_GetError (QSPI_HandleTypeDef * hqspi)

Function description

Return the QSPI error code.

Parameters

- **hqspi:** : QSPI handle

Return values

- **QSPI:** Error Code

HAL_QSPI_Abort

Function name

HAL_StatusTypeDef HAL_QSPI_Abort (QSPI_HandleTypeDef * hqspi)

Function description

Abort the current transmission.

Parameters

- **hqspi:** : QSPI handle

Return values

- **HAL:** status

HAL_QSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Abort_IT (QSPI_HandleTypeDef * hqspi)

Function description

Abort the current transmission (non-blocking function)

Parameters

- **hqspi:** : QSPI handle

Return values

- **HAL:** status

HAL_QSPI_SetTimeout

Function name

void HAL_QSPI_SetTimeout (QSPI_HandleTypeDef * hqspi, uint32_t Timeout)

Function description

Set QSPI timeout.

Parameters

- **hqspi:** : QSPI handle.
- **Timeout:** : Timeout for the QSPI memory access.

Return values

- **None:**

HAL_QSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_QSPI_SetFifoThreshold (QSPI_HandleTypeDef * hqspi, uint32_t Threshold)

Function description

Set QSPI Fifo threshold.

Parameters

- **hqspi:** : QSPI handle.
- **Threshold:** : Threshold of the Fifo (value between 1 and 16).

Return values

- **HAL:** status

HAL_QSPI_GetFifoThreshold

Function name

uint32_t HAL_QSPI_GetFifoThreshold (QSPI_HandleTypeDef * hqspi)

Function description

Get QSPI Fifo threshold.

Parameters

- **hqspi:** : QSPI handle.

Return values

- **Fifo:** threshold (value between 1 and 16)

HAL_QSPI_SetFlashID

Function name

HAL_StatusTypeDef HAL_QSPI_SetFlashID (QSPI_HandleTypeDef * hqspi, uint32_t FlashID)

Function description

Set FlashID.

Parameters

- **hqspi:** : QSPI handle.
- **FlashID:** : Index of the flash memory to be accessed. This parameter can be a value of QSPI Flash Select.

Return values

- **HAL:** status

Notes

- The FlashID is ignored when dual flash mode is enabled.

44.3 QSPI Firmware driver defines

The following section lists the various define and macros of the module.

44.3.1 QSPI

QSPI

QSPI Address Mode

QSPI_ADDRESS_NONE

No address

QSPI_ADDRESS_1_LINE

Address on a single line

QSPI_ADDRESS_2_LINES

Address on two lines

QSPI_ADDRESS_4_LINES

Address on four lines

QSPI Address Size

QSPI_ADDRESS_8_BITS

8-bit address

QSPI_ADDRESS_16_BITS

16-bit address

QSPI_ADDRESS_24_BITS

24-bit address

QSPI_ADDRESS_32_BITS

32-bit address

QSPI Alternate Bytes Mode

QSPI_ALTERNATE_BYTES_NONE

No alternate bytes

QSPI_ALTERNATE_BYTES_1_LINE

Alternate bytes on a single line

QSPI_ALTERNATE_BYTES_2_LINES

Alternate bytes on two lines

QSPI_ALTERNATE_BYTES_4_LINES

Alternate bytes on four lines

QSPI Alternate Bytes Size

QSPI_ALTERNATE_BYTES_8_BITS

8-bit alternate bytes

QSPI_ALTERNATE_BYTES_16_BITS

16-bit alternate bytes

QSPI_ALTERNATE_BYTES_24_BITS

24-bit alternate bytes

QSPI_ALTERNATE_BYTES_32_BITS

32-bit alternate bytes

QSPI Automatic Stop

QSPI_AUTOMATIC_STOP_DISABLE

AutoPolling stops only with abort or QSPI disabling

QSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

QSPI ChipSelect High Time

QSPI_CS_HIGH_TIME_1_CYCLE

nCS stay high for at least 1 clock cycle between commands

QSPI_CS_HIGH_TIME_2_CYCLE

nCS stay high for at least 2 clock cycles between commands

QSPI_CS_HIGH_TIME_3_CYCLE

nCS stay high for at least 3 clock cycles between commands

QSPI_CS_HIGH_TIME_4_CYCLE

nCS stay high for at least 4 clock cycles between commands

QSPI_CS_HIGH_TIME_5_CYCLE

nCS stay high for at least 5 clock cycles between commands

QSPI_CS_HIGH_TIME_6_CYCLE

nCS stay high for at least 6 clock cycles between commands

QSPI_CS_HIGH_TIME_7_CYCLE

nCS stay high for at least 7 clock cycles between commands

QSPI_CS_HIGH_TIME_8_CYCLE

nCS stay high for at least 8 clock cycles between commands

QSPI Clock Mode

QSPI_CLOCK_MODE_0

Clk stays low while nCS is released

QSPI_CLOCK_MODE_3

Clk goes high while nCS is released

QSPI Data Mode

QSPI_DATA_NONE

No data

QSPI_DATA_1_LINE

Data on a single line

QSPI_DATA_2_LINES

Data on two lines

QSPI_DATA_4_LINES

Data on four lines

QSPI DDR Data Output Delay

QSPI_DDR_HHC_ANALOG_DELAY

Delay the data output using analog delay in DDR mode

QSPI_DDR_HHC_HALF_CLK_DELAY

Delay the data output by one quarter of QUADSPI output clock in DDR mode

QSPI DDR Mode

QSPI_DDR_MODE_DISABLE

Double data rate mode disabled

QSPI_DDR_MODE_ENABLE

Double data rate mode enabled

QSPI Dual Flash Mode

QSPI_DUALFLASH_ENABLE

Dual-flash mode enabled

QSPI_DUALFLASH_DISABLE

Dual-flash mode disabled

QSPI Error Code

HAL_QSPI_ERROR_NONE

No error

HAL_QSPI_ERROR_TIMEOUT

Timeout error

HAL_QSPI_ERROR_TRANSFER

Transfer error

HAL_QSPI_ERROR_DMA

DMA transfer error

HAL_QSPI_ERROR_INVALID_PARAM

Invalid parameters error

QSPI Exported Macros

__HAL_QSPI_RESET_HANDLE_STATE

Description:

- Reset QSPI handle state.

Parameters:

- __HANDLE__: : QSPI handle.

Return value:

- None

__HAL_QSPI_ENABLE

Description:

- Enable the QSPI peripheral.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_DISABLE

Description:

- Disable the QSPI peripheral.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_ENABLE_IT

Description:

- Enable the specified QSPI interrupt.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.
- __INTERRUPT__: : specifies the QSPI interrupt source to enable. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

__HAL_QSPI_DISABLE_IT

Description:

- Disable the specified QSPI interrupt.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.
- __INTERRUPT__: : specifies the QSPI interrupt source to disable. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

__HAL_QSPI_GET_IT_SOURCE

Description:

- Check whether the specified QSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.
- __INTERRUPT__: : specifies the QSPI interrupt source to check. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_QSPI_GET_FLAG

Description:

- Check whether the selected QSPI flag is set or not.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.
- __FLAG__: : specifies the QSPI flag to check. This parameter can be one of the following values:
  - QSPI_FLAG_BUSY: QSPI Busy flag
  - QSPI_FLAG_TO: QSPI Timeout flag
  - QSPI_FLAG_SM: QSPI Status match flag
  - QSPI_FLAG_FT: QSPI FIFO threshold flag
  - QSPI_FLAG_TC: QSPI Transfer complete flag
  - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

__HAL_QSPI_CLEAR_FLAG

Description:

- Clears the specified QSPI's flag status.

Parameters:

- __HANDLE__: : specifies the QSPI Handle.
- __FLAG__: : specifies the QSPI clear register flag that needs to be set This parameter can be one of the following values:
  - QSPI_FLAG_TO: QSPI Timeout flag
  - QSPI_FLAG_SM: QSPI Status match flag
  - QSPI_FLAG_TC: QSPI Transfer complete flag
  - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

QSPI Flags

QSPI_FLAG_BUSY

Busy flag: operation is ongoing

QSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

QSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

QSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

QSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

QSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

QSPI Flash Select

QSPI_FLASH_ID_1

FLASH 1 selected

QSPI_FLASH_ID_2

FLASH 2 selected

QSPI Instruction Mode

QSPI_INSTRUCTION_NONE

No instruction

QSPI_INSTRUCTION_1_LINE

Instruction on a single line

QSPI_INSTRUCTION_2_LINES

Instruction on two lines

QSPI_INSTRUCTION_4_LINES

Instruction on four lines

QSPI Interrupts

QSPI_IT_TO

Interrupt on the timeout flag

QSPI_IT_SM

Interrupt on the status match flag

QSPI_IT_FT

Interrupt on the fifo threshold flag

QSPI_IT_TC

Interrupt on the transfer complete flag

QSPI_IT_TE

Interrupt on the transfer error flag

QSPI Match Mode

QSPI_MATCH_MODE_AND

AND match mode between unmasked bits

QSPI_MATCH_MODE_OR

OR match mode between unmasked bits

QSPI Sample Shifting

QSPI_SAMPLE_SHIFTING_NONE

No clock cycle shift to sample data

QSPI_SAMPLE_SHIFTING_HALFCYCLE

1/2 clock cycle shift to sample data

QSPI Send Instruction Mode

QSPI_SIOO_INST_EVERY_CMD

Send instruction on every transaction

QSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

QSPI Timeout Activation

QSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

QSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

QSPI Timeout definition

HAL_QSPI_TIMEOUT_DEFAULT_VALUE

45 HAL RCC Generic Driver

45.1 RCC Firmware driver registers structures

45.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32g4xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t PLLQ***
- ***uint32_t PLLR***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  The new state of the PLL. This parameter can be a value of ***RCC_PLL_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
  RCC_PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLM***
  PLLM: Division factor for PLL VCO input clock. This parameter must be a value of ***RCC_PLLM_Clock_Divider***
- ***uint32_t RCC_PLLInitTypeDef::PLLN***
  PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 8 and Max_Data = 127
- ***uint32_t RCC_PLLInitTypeDef::PLLP***
  PLLP: Division factor for ADC clock. This parameter must be a value of ***RCC_PLLP_Clock_Divider***
- ***uint32_t RCC_PLLInitTypeDef::PLLQ***
  PLLQ: Division factor for SAI, I2S, USB, FDCAN and QUADSPI clocks. This parameter must be a value of ***RCC_PLLQ_Clock_Divider***
- ***uint32_t RCC_PLLInitTypeDef::PLLR***
  PLLR: Division for the main system clock. User have to set the PLLR parameter correctly to not exceed max frequency 170MHZ. This parameter must be a value of ***RCC_PLLR_Clock_Divider***

45.1.2 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32g4xx_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t HSI48State***
- ***RCC_PLLInitTypeDef PLL***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***

- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSI48State***
  The new state of the HSI48. This parameter can be a value of ***RCC_HSI48_Config***
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
  Main PLL structure parameters

45.1.3 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32g4xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source used as system clock (SYSCLK). This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***

45.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

45.2.1 RCC specific features

After reset the device is running from High Speed Internal oscillator (16 MHz) with Flash 0 wait state. Flash prefetch buffer, D-Cache and I-Cache are disabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHBs) and Low speed (APBs) busses: all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analog mode, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals which clocks are not derived from the System clock (USB, RNG, USART, LPUART, FDCAN, some TIMERs, UCPD, I2S, I2C, LPTIM, ADC, QSPI)

45.2.2 Initialization and de-initialization functions

This section provides functions allowing to configure the internal and external oscillators (HSE, HSI, LSE, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

- HSI (high-speed internal): 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
- LSI (low-speed internal): 32 KHz low consumption RC used as IWDG and/or RTC clock source.
- HSE (high-speed external): 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also optionally as RTC clock source.
- LSE (low-speed external): 32.768 KHz oscillator used optionally as RTC clock source.
- PLL (clocked by HSI, HSE) providing up to three independent output clocks:
  - The first output is used to generate the high speed system clock (up to 170 MHz).
  - The second output is used to generate the clock for the USB (48 MHz), the QSPI (<= 48 MHz), the FDCAN, the SAI and the I2S.
  - The third output is used to generate a clock for ADC
- CSS (Clock security system): once enabled, if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
- MCO (microcontroller clock output): used to output LSI, HSI, LSE, HSE, main PLL clock, system clock or RC48 clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

- Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and main PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks.

Note: All the peripheral clocks are derived from the System clock (SYSCLK) except:

- *RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use __HAL_RCC_RTC_ENABLE() and HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *USB FS and RNG: USB FS requires a frequency equal to 48 MHz to work correctly, while the RNG peripheral requires a frequency equal or lower than to 48 MHz. This clock is derived of the main PLL through PLLQ divider. You have to enable the peripheral clock and use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *IWDG clock which is always the LSI clock.*
- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 170 MHz. The clock source frequency should be adapted depending on the device voltage range as listed in the Reference Manual "Clock source frequency versus voltage scaling" chapter.

This section contains the following APIs:

- ***HAL_RCC_DeInit***
- ***HAL_RCC_OscConfig***
- ***HAL_RCC_ClockConfig***

45.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to:

- Ouput clock to MCO pin.
- Retrieve current clock frequencies.
- Enable the Clock Security System.

This section contains the following APIs:

- ***HAL_RCC_MCOConfig***
- ***HAL_RCC_GetSysClockFreq***
- ***HAL_RCC_GetHCLKFreq***
- ***HAL_RCC_GetPCLK1Freq***
- ***HAL_RCC_GetPCLK2Freq***
- ***HAL_RCC_GetOscConfig***
- ***HAL_RCC_GetClockConfig***
- ***HAL_RCC_EnableCSS***
- ***HAL_RCC_EnableLSECSS***
- ***HAL_RCC_DisableLSECSS***
- ***HAL_RCC_NMI_IRQHandler***
- ***HAL_RCC_CSSCallback***

45.2.4 Detailed description of functions

HAL_RCC_DeInit

Function name

HAL_StatusTypeDef HAL_RCC_DeInit (void)

Function description

Reset the RCC clock configuration to the default reset state.

Return values

- **HAL:** status

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO1 OFFAll interrupts disabledAll interrupt and reset flags cleared
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Initialize the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- The PLL is not disabled when used as system clock.
- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)

Function description

Initialize the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.
- **FLatency:** FLASH Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0 FLASH 0 Latency cycle
  - FLASH_LATENCY_1 FLASH 1 Latency cycle
  - FLASH_LATENCY_2 FLASH 2 Latency cycles
  - FLASH_LATENCY_3 FLASH 3 Latency cycles
  - FLASH_LATENCY_4 FLASH 4 Latency cycles
  - FLASH_LATENCY_5 FLASH 5 Latency cycles
  - FLASH_LATENCY_6 FLASH 6 Latency cycles
  - FLASH_LATENCY_7 FLASH 7 Latency cycles
  - FLASH_LATENCY_8 FLASH 8 Latency cycles
  - FLASH_LATENCY_9 FLASH 9 Latency cycles
  - FLASH_LATENCY_10 FLASH 10 Latency cycles
  - FLASH_LATENCY_11 FLASH 11 Latency cycles
  - FLASH_LATENCY_12 FLASH 12 Latency cycles
  - FLASH_LATENCY_13 FLASH 13 Latency cycles
  - FLASH_LATENCY_14 FLASH 14 Latency cycles
  - FLASH_LATENCY_15 FLASH 15 Latency cycles

Return values

- **None:**

Notes

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function
- The HSI is used by default as system clock source after startup from Reset, wake-up from STANDBY mode. After restart from Reset, the HSI frequency is set to its default value 16 MHz.
- The HSI can be selected as system clock source after from STOP modes or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source is ready.
- You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.
- Depending on the device voltage range, the software has to set correctly HPRE[3:0] bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

HAL_RCC_MCOConfig

Function name

void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)

Function description

Select the clock source to output on MCO pin(PA8).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. For STM32G4xx family this parameter can have only one value:
  - RCC_MCO1 Clock source to output on MCO1 pin(PA8).
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK MCO output disabled, no clock on MCO
  - RCC_MCO1SOURCE_SYSCLK system clock selected as MCO source
  - RCC_MCO1SOURCE_HSI HSI clock selected as MCO source
  - RCC_MCO1SOURCE_HSE HSE clock selected as MCO sourcee
  - RCC_MCO1SOURCE_PLLCLK main PLL clock selected as MCO source
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO source
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO source
  - RCC_MCO1SOURCE_HSI48 HSI48 clock selected as MCO source for devices with HSI48
- **RCC_MCODiv:** specifies the MCO prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 no division applied to MCO clock
  - RCC_MCODIV_2 division by 2 applied to MCO clock
  - RCC_MCODIV_4 division by 4 applied to MCO clock
  - RCC_MCODIV_8 division by 8 applied to MCO clock
  - RCC_MCODIV_16 division by 16 applied to MCO clock

Return values

- **None:**

Notes

- PA8 should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name

void HAL_RCC_EnableCSS (void)

Function description

Enable the Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
- The Clock Security System can only be cleared by reset.

HAL_RCC_EnableLSECSS

Function name

void HAL_RCC_EnableLSECSS (void)

Function description

Enable the LSE Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the external 32 kHz oscillator, the LSE clock is no longer supplied to the RTC but no hardware action is made to the registers. If enabled, an interrupt will be generated and handle through RCC LSE CSS external interrupt line
- The Clock Security System can only be cleared by reset or after a LSE failure detection.

HAL_RCC_DisableLSECSS

Function name

void HAL_RCC_DisableLSECSS (void)

Function description

Disable the LSE Clock Security System.

Return values

- **None:**

Notes

- After LSE failure detection, the software must disable LSECSSON
- The Clock Security System can only be cleared by reset otherwise.

HAL_RCC_GetSysClockFreq

Function name

uint32_t HAL_RCC_GetSysClockFreq (void)

Function description

Return the SYSCLK frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)
- If SYSCLK source is PLL, function returns values based on HSE_VALUE(**), HSI_VALUE(*) Value multiplied/divided by the PLL factors.
- (*) HSI_VALUE is a constant defined in stm32g4xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) HSE_VALUE is a constant defined in stm32g4xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name

uint32_t HAL_RCC_GetHCLKFreq (void)

Function description

Return the HCLK frequency.

Return values

- **HCLK:** frequency in Hz

Notes

- Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency.

HAL_RCC_GetPCLK1Freq

Function name

uint32_t HAL_RCC_GetPCLK1Freq (void)

Function description

Return the PCLK1 frequency.

Return values

- **PCLK1:** frequency in Hz

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name

uint32_t HAL_RCC_GetPCLK2Freq (void)

Function description

Return the PCLK2 frequency.

Return values

- **PCLK2:** frequency in Hz

Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name

void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Configure the RCC_OscInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.

Return values

- **None:**

HAL_RCC_GetClockConfig

Function name

void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)

Function description

Configure the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that will be configured.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

HAL_RCC_NMI_IRQHandler

Function name

void HAL_RCC_NMI_IRQHandler (void)

Function description

Handle the RCC Clock Security System interrupt request.

Return values

- **None:**

Notes

- This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name

void HAL_RCC_CSSCallback (void)

Function description

RCC Clock Security System interrupt callback.

Return values

- **none:**

45.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

45.3.1 RCC

RCC

AHB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_DMA1_CLK_SLEEP_ENABLE

__HAL_RCC_DMA2_CLK_SLEEP_ENABLE

__HAL_RCC_DMAMUX1_CLK_SLEEP_ENABLE

__HAL_RCC_CORDIC_CLK_SLEEP_ENABLE

__HAL_RCC_FMAC_CLK_SLEEP_ENABLE

__HAL_RCC_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_DMA1_CLK_SLEEP_DISABLE

__HAL_RCC_DMA2_CLK_SLEEP_DISABLE

__HAL_RCC_DMAMUX1_CLK_SLEEP_DISABLE

__HAL_RCC_CORDIC_CLK_SLEEP_DISABLE

__HAL_RCC_FMAC_CLK_SLEEP_DISABLE

__HAL_RCC_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

AHB1 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMAMUX1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CORDIC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FMAC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SRAM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMAMUX1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CORDIC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FMAC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SRAM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_DISABLED

AHB1 Peripheral Force Release Reset

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_DMA1_FORCE_RESET

__HAL_RCC_DMA2_FORCE_RESET

__HAL_RCC_DMAMUX1_FORCE_RESET

__HAL_RCC_CORDIC_FORCE_RESET

__HAL_RCC_FMAC_FORCE_RESET

__HAL_RCC_FLASH_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_DMA1_RELEASE_RESET

__HAL_RCC_DMA2_RELEASE_RESET

__HAL_RCC_DMAMUX1_RELEASE_RESET

__HAL_RCC_CORDIC_RELEASE_RESET

__HAL_RCC_FMAC_RELEASE_RESET

__HAL_RCC_FLASH_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

AHB1 Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE

__HAL_RCC_DMA2_CLK_ENABLE

__HAL_RCC_DMAMUX1_CLK_ENABLE

__HAL_RCC_CORDIC_CLK_ENABLE

__HAL_RCC_FMAC_CLK_ENABLE

__HAL_RCC_FLASH_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_DMA2_CLK_DISABLE

__HAL_RCC_DMAMUX1_CLK_DISABLE

__HAL_RCC_CORDIC_CLK_DISABLE

__HAL_RCC_FMAC_CLK_DISABLE

__HAL_RCC_FLASH_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

AHB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_ENABLED

__HAL_RCC_DMA2_IS_CLK_ENABLED

__HAL_RCC_DMAMUX1_IS_CLK_ENABLED

__HAL_RCC_CORDIC_IS_CLK_ENABLED

__HAL_RCC_FMAC_IS_CLK_ENABLED

__HAL_RCC_FLASH_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_DMA2_IS_CLK_DISABLED

__HAL_RCC_DMAMUX1_IS_CLK_DISABLED

__HAL_RCC_CORDIC_IS_CLK_DISABLED

__HAL_RCC_FMAC_IS_CLK_DISABLED

__HAL_RCC_FLASH_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

AHB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_ADC12_IS_CLK_ENABLED

__HAL_RCC_ADC345_IS_CLK_ENABLED

__HAL_RCC_DAC1_IS_CLK_ENABLED

__HAL_RCC_DAC2_IS_CLK_ENABLED

__HAL_RCC_DAC3_IS_CLK_ENABLED

__HAL_RCC_DAC4_IS_CLK_ENABLED

__HAL_RCC_AES_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_ADC12_IS_CLK_DISABLED

__HAL_RCC_ADC345_IS_CLK_DISABLED

__HAL_RCC_DAC1_IS_CLK_DISABLED

__HAL_RCC_DAC2_IS_CLK_DISABLED

__HAL_RCC_DAC3_IS_CLK_DISABLED

__HAL_RCC_DAC4_IS_CLK_DISABLED

__HAL_RCC_AES_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

AHB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_CCM_CLK_SLEEP_ENABLE

__HAL_RCC_ADC12_CLK_SLEEP_ENABLE

__HAL_RCC_ADC345_CLK_SLEEP_ENABLE

__HAL_RCC_DAC1_CLK_SLEEP_ENABLE

__HAL_RCC_DAC2_CLK_SLEEP_ENABLE

__HAL_RCC_DAC3_CLK_SLEEP_ENABLE

__HAL_RCC_DAC4_CLK_SLEEP_ENABLE

__HAL_RCC_AES_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_CCM_CLK_SLEEP_DISABLE

__HAL_RCC_ADC12_CLK_SLEEP_DISABLE

__HAL_RCC_ADC345_CLK_SLEEP_DISABLE

__HAL_RCC_DAC1_CLK_SLEEP_DISABLE

__HAL_RCC_DAC2_CLK_SLEEP_DISABLE

__HAL_RCC_DAC3_CLK_SLEEP_DISABLE

__HAL_RCC_DAC4_CLK_SLEEP_DISABLE

__HAL_RCC_AES_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

AHB2 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SRAM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CCM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC345_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_AES_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SRAM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CCM_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC12_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC345_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_AES_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_DISABLED

AHB2 Peripheral Force Release Reset

__HAL_RCC_AHB2_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_ADC12_FORCE_RESET

__HAL_RCC_ADC345_FORCE_RESET

__HAL_RCC_DAC1_FORCE_RESET

__HAL_RCC_DAC2_FORCE_RESET

__HAL_RCC_DAC3_FORCE_RESET

__HAL_RCC_DAC4_FORCE_RESET

__HAL_RCC_AES_FORCE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_AHB2_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_ADC12_RELEASE_RESET

__HAL_RCC_ADC345_RELEASE_RESET

__HAL_RCC_DAC1_RELEASE_RESET

__HAL_RCC_DAC2_RELEASE_RESET

__HAL_RCC_DAC3_RELEASE_RESET

__HAL_RCC_DAC4_RELEASE_RESET

__HAL_RCC_AES_RELEASE_RESET

__HAL_RCC_RNG_RELEASE_RESET

AHB2 Peripheral Clock Enable Disable

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_ADC12_CLK_ENABLE

__HAL_RCC_ADC345_CLK_ENABLE

__HAL_RCC_DAC1_CLK_ENABLE

__HAL_RCC_DAC2_CLK_ENABLE

__HAL_RCC_DAC3_CLK_ENABLE

__HAL_RCC_DAC4_CLK_ENABLE

__HAL_RCC_AES_CLK_ENABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_ADC12_CLK_DISABLE

__HAL_RCC_ADC345_CLK_DISABLE

__HAL_RCC_DAC1_CLK_DISABLE

__HAL_RCC_DAC2_CLK_DISABLE

__HAL_RCC_DAC3_CLK_DISABLE

__HAL_RCC_DAC4_CLK_DISABLE

__HAL_RCC_AES_CLK_DISABLE

__HAL_RCC_RNG_CLK_DISABLE

AHB3 Peripheral Clock Enable Disable

__HAL_RCC_FMC_CLK_ENABLE

__HAL_RCC_QSPI_CLK_ENABLE

__HAL_RCC_FMC_CLK_DISABLE

__HAL_RCC_QSPI_CLK_DISABLE

AHB3 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_FMC_IS_CLK_ENABLED

__HAL_RCC_QSPI_IS_CLK_ENABLED

__HAL_RCC_FMC_IS_CLK_DISABLED

__HAL_RCC_QSPI_IS_CLK_DISABLED

AHB3 Peripheral Clock Sleep Enable Disable

__HAL_RCC_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_QSPI_CLK_SLEEP_ENABLE

__HAL_RCC_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_QSPI_CLK_SLEEP_DISABLE

AHB3 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_FMC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_QSPI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FMC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_QSPI_IS_CLK_SLEEP_DISABLED

AHB3 Peripheral Force Release Reset

__HAL_RCC_AHB3_FORCE_RESET

__HAL_RCC_FMC_FORCE_RESET

__HAL_RCC_QSPI_FORCE_RESET

__HAL_RCC_AHB3_RELEASE_RESET

__HAL_RCC_FMC_RELEASE_RESET

__HAL_RCC_QSPI_RELEASE_RESET

AHB Clock Source

RCC_SYSCLK_DIV1

SYSCLK not divided

RCC_SYSCLK_DIV2

SYSCLK divided by 2

RCC_SYSCLK_DIV4

SYSCLK divided by 4

RCC_SYSCLK_DIV8

SYSCLK divided by 8

RCC_SYSCLK_DIV16

SYSCLK divided by 16

RCC_SYSCLK_DIV64

SYSCLK divided by 64

RCC_SYSCLK_DIV128

SYSCLK divided by 128

RCC_SYSCLK_DIV256

SYSCLK divided by 256

RCC_SYSCLK_DIV512

SYSCLK divided by 512

APB1 APB2 Clock Source

RCC_HCLK_DIV1

HCLK not divided

RCC_HCLK_DIV2

HCLK divided by 2

RCC_HCLK_DIV4

HCLK divided by 4

RCC_HCLK_DIV8

HCLK divided by 8

RCC_HCLK_DIV16

HCLK divided by 16

APB1 Peripheral Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_CRS_CLK_ENABLE

__HAL_RCC_RTCAPB_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_USB_CLK_ENABLE

__HAL_RCC_FDCAN_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_I2C4_CLK_ENABLE

__HAL_RCC_UCPD1_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_CRS_CLK_DISABLE

__HAL_RCC_RTCAPB_CLK_DISABLE

__HAL_RCC_WWDG_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_USB_CLK_DISABLE

__HAL_RCC_FDCAN_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_I2C4_CLK_DISABLE

__HAL_RCC_UCPD1_CLK_DISABLE

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_CRS_IS_CLK_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_USB_IS_CLK_ENABLED

__HAL_RCC_FDCAN_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_I2C4_IS_CLK_ENABLED

__HAL_RCC_UCPD1_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_CRS_IS_CLK_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_DISABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_USB_IS_CLK_DISABLED

__HAL_RCC_FDCAN_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

__HAL_RCC_I2C3_IS_CLK_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_I2C4_IS_CLK_DISABLED

__HAL_RCC_UCPD1_IS_CLK_DISABLED

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_USB_CLK_SLEEP_ENABLE

__HAL_RCC_FDCAN_CLK_SLEEP_ENABLE

__HAL_RCC_PWR_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_UCPD1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_USB_CLK_SLEEP_DISABLE

__HAL_RCC_FDCAN_CLK_SLEEP_DISABLE

__HAL_RCC_PWR_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_UCPD1_CLK_SLEEP_DISABLE

APB1 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FDCAN_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UCPD1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FDCAN_IS_CLK_SLEEP_DISABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UCPD1_IS_CLK_SLEEP_DISABLED

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_CRS_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_USB_FORCE_RESET

__HAL_RCC_FDCAN_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_I2C4_FORCE_RESET

__HAL_RCC_UCPD1_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_CRS_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_USB_RELEASE_RESET

__HAL_RCC_FDCAN_RELEASE_RESET

__HAL_RCC_PWR_RELEASE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_I2C4_RELEASE_RESET

__HAL_RCC_UCPD1_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_SPI4_CLK_ENABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_TIM20_CLK_ENABLE

__HAL_RCC_SAI1_CLK_ENABLE

__HAL_RCC_HRTIM1_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_SPI4_CLK_DISABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_TIM20_CLK_DISABLE

__HAL_RCC_SAI1_CLK_DISABLE

__HAL_RCC_HRTIM1_CLK_DISABLE

APB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_SPI4_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_TIM20_IS_CLK_ENABLED

__HAL_RCC_SAI1_IS_CLK_ENABLED

__HAL_RCC_HRTIM1_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_SPI4_IS_CLK_DISABLED

__HAL_RCC_TIM15_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

__HAL_RCC_TIM20_IS_CLK_DISABLED

__HAL_RCC_SAI1_IS_CLK_DISABLED

__HAL_RCC_HRTIM1_IS_CLK_DISABLED

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_TIM20_CLK_SLEEP_ENABLE

__HAL_RCC_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_HRTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_TIM20_CLK_SLEEP_DISABLE

__HAL_RCC_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_HRTIM1_CLK_SLEEP_DISABLE

APB2 Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM8_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM20_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HRTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM8_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM20_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_HRTIM1_IS_CLK_SLEEP_DISABLED

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_SPI4_FORCE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_TIM20_FORCE_RESET

__HAL_RCC_SAI1_FORCE_RESET

__HAL_RCC_HRTIM1_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_SPI4_RELEASE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_TIM20_RELEASE_RESET

__HAL_RCC_SAI1_RELEASE_RESET

__HAL_RCC_HRTIM1_RELEASE_RESET

RCC Backup Domain Reset

__HAL_RCC_BACKUPRESET_FORCE

Description:

- Macros to force or release the Backup domain reset.

Return value:

- None

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

RCC Exported Macros

__HAL_RCC_HSI_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 16MHz oscillator (HSI).

Return value:

- None

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed 16MHz oscillator (HSI) calibration value.

Parameters:

- __HSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x7F.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

__HAL_RCC_HSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USARTs and I2Cs.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI startup time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSISTOP_DISABLE

__HAL_RCC_LSI_ENABLE

Description:

- Macros to enable or disable the Internal Low Speed oscillator (LSI).

Return value:

- None

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC_HSE_OFF Turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC_HSE_ON Turn ON the HSE oscillator.
  - RCC_HSE_BYPASS HSE oscillator bypassed with external clock.

Return value:

- None

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

__HAL_RCC_LSE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC_LSE_OFF Turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC_LSE_ON Turn ON the LSE oscillator.
  - RCC_LSE_BYPASS LSE oscillator bypassed with external clock.

Return value:

- None

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

__HAL_RCC_HSI48_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 48MHz oscillator (HSI48).

Return value:

- None

Notes:

- The HSI48 is stopped by hardware when entering STOP and STANDBY modes. After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSI48_DISABLE

__HAL_RCC_RTC_CONFIG

Description:

- Macros to configure the RTC clock (RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_NONE No clock selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32 selected

Return value:

- None

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the Backup domain is reset using __HAL_RCC_BACKUPRESET_FORCE() macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_GET_RTC_SOURCE

Description:

- Macro to get the RTC clock source.

Return value:

- The: returned value can be one of the following:
  - RCC_RTCCLKSOURCE_NONE No clock selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32 selected

__HAL_RCC_PLL_ENABLE

Description:

- Macros to enable or disable the main PLL.

Return value:

- None

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

__HAL_RCC_PLL_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL clock source.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_NONE No clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.

__HAL_RCC_PLL_PLLM_CONFIG

Description:

- Macro to configure the PLL source division factor M.

Parameters:

- __PLLM__: specifies the division factor for PLL VCO input clock This parameter must be a value of

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 2.66 to 8 MHz. It is recommended to select a frequency of 8 MHz to limit PLL jitter.

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configure the main PLL clock source, multiplication and division factors.

Parameters:

- __PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_NONE No clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL clock entry
- __PLLM__: specifies the division factor for PLL VCO input clock. This parameter must be a value of
- __PLLN__: specifies the multiplication factor for PLL VCO output clock. This parameter must be a number between 8 and 127.
- __PLLP__: specifies the division factor for SAI clock. This parameter must be a number in the range (2 to 31).
- __PLLQ__: specifies the division factor for OTG FS, SDMMC1 and RNG clocks. This parameter must be in the range (2, 4, 6 or 8).
- __PLLR__: specifies the division factor for the main system clock.

Return value:

- None

Notes:

- This macro must be used only when the main PLL is disabled. This macro preserves the PLL's output clocks enable state.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 2.66 to 8 MHz. It is recommended to select a frequency of 8 MHz to limit PLL jitter.
- You have to set the PLLN parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz.
- If the USB OTG FS is used in your application, you have to set the PLLQ parameter correctly to have 48 MHz clock for the USB. However, the SDMMC1 and RNG need a frequency lower than or equal to 48 MHz to work correctly.
- You have to set the PLLR parameter correctly to not exceed 170MHZ. This parameter must be in the range (2, 4, 6 or 8).

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_PLLCLKOUT_ENABLE

Description:

- Enable or disable each clock output (RCC_PLL_SYSCLK, RCC_PLL_48M1CLK, RCC_PLL_ADCCLK)

Parameters:

- __PLLCLOCKOUT__: specifies the PLL clock to be output. This parameter can be one or a combination of the following values:
  - RCC_PLL_ADCCLK This clock is used to generate a clock on ADC.
  - RCC_PLL_48M1CLK This Clock is used to generate the clock for the USB (48 MHz), FDCAN (<=48 MHz) and QSPI (<=48 MHz).
  - RCC_PLL_SYSCLK This Clock is used to generate the high speed system clock (up to 170MHz)

Return value:

- None

Notes:

- Enabling/disabling clock outputs RCC_PLL_ADCCLK and RCC_PLL_48M1CLK can be done at anytime without the need to stop the PLL in order to save power. But RCC_PLL_SYSCLK cannot be stopped if used as System Clock.

__HAL_RCC_PLLCLKOUT_DISABLE

__HAL_RCC_GET_PLLCLKOUT_CONFIG

Description:

- Get clock output enable status (RCC_PLL_SYSCLK, RCC_PLL_48M1CLK, RCC_PLL_SAI3CLK)

Parameters:

- __PLLCLOCKOUT__: specifies the output PLL clock to be checked. This parameter can be one of the following values:
  - RCC_PLL_ADCCLK This clock is used to generate a clock on ADC.
  - RCC_PLL_48M1CLK This Clock is used to generate the clock for the USB (48 MHz), FDCAN (<=48 MHz) and QSPI (<=48 MHz).
  - RCC_PLL_SYSCLK This Clock is used to generate the high speed system clock (up to 170MHz)

Return value:

- SET: / RESET

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_PLLCLK: PLL output is used as system clock source.

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_PLLCLK: PLL used as system clock.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - RCC_LSEDRIVE_LOW LSE oscillator low drive capability.
  - RCC_LSEDRIVE_MEDIUMLOW LSE oscillator medium low drive capability.
  - RCC_LSEDRIVE_MEDIUMHIGH LSE oscillator medium high drive capability.
  - RCC_LSEDRIVE_HIGH LSE oscillator high drive capability.

Return value:

- None

Notes:

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset).

__HAL_RCC_MCO1_CONFIG

Description:

- Macro to configure the MCO clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK MCO output disabled
  - RCC_MCO1SOURCE_SYSCLK System clock selected as MCO source
  - RCC_MCO1SOURCE_HSI HSI clock selected as MCO source
  - RCC_MCO1SOURCE_HSE HSE clock selected as MCO sourcee
  - RCC_MCO1SOURCE_PLLCLK Main PLL clock selected as MCO source
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO source
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO source
  - RCC_MCO1SOURCE_HSI48 HSI48 clock selected as MCO source for devices with HSI48
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 MCO clock source is divided by 1
  - RCC_MCODIV_2 MCO clock source is divided by 2
  - RCC_MCODIV_4 MCO clock source is divided by 4
  - RCC_MCODIV_8 MCO clock source is divided by 8
  - RCC_MCODIV_16 MCO clock source is divided by 16

Flags

RCC_FLAG_HSIRDY

HSI Ready flag

RCC_FLAG_HSERDY

HSE Ready flag

RCC_FLAG_PLLRDY

PLL Ready flag

RCC_FLAG_LSERDY

LSE Ready flag

RCC_FLAG_LSECSSD

LSE Clock Security System Interrupt flag

RCC_FLAG_LSIRDY

LSI Ready flag

RCC_FLAG_OBLRST

Option Byte Loader reset flag

RCC_FLAG_PINRST

PIN reset flag

RCC_FLAG_BORRST

BOR reset flag

RCC_FLAG_SFTRST

Software Reset flag

RCC_FLAG_IWDGRST

Independent Watchdog reset flag

RCC_FLAG_WWDGRST

Window watchdog reset flag

RCC_FLAG_LPWRRST

Low-Power reset flag

RCC_FLAG_HSI48RDY

HSI48 Ready flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to enable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48

Return value:

- None

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to disable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48

Return value:

- None

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CIR[23:16] bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48

Return value:

- None

__HAL_RCC_GET_IT

Description:

- Check whether the RCC interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

Description:

- Set RMVF bit to clear the reset flags.

Return value:

- None

__HAL_RCC_GET_FLAG

Description:

- Check whether the selected RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_HSIRDY HSI oscillator clock ready
  - RCC_FLAG_HSERDY HSE oscillator clock ready
  - RCC_FLAG_PLLRDY Main PLL clock ready
  - RCC_FLAG_HSI48RDY HSI48 clock ready for devices with HSI48
  - RCC_FLAG_LSERDY LSE oscillator clock ready
  - RCC_FLAG_LSECSSD Clock security system failure on LSE oscillator detection
  - RCC_FLAG_LSIRDY LSI oscillator clock ready
  - RCC_FLAG_BORRST BOR reset
  - RCC_FLAG_OBLRST OBLRST reset
  - RCC_FLAG_PINRST Pin reset
  - RCC_FLAG_SFTRST Software reset
  - RCC_FLAG_IWDGRST Independent Watchdog reset
  - RCC_FLAG_WWDGRST Window Watchdog reset
  - RCC_FLAG_LPWRRST Low Power reset

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

RCC_HSE_BYPASS

External clock source for HSE clock

HSI48 Config

RCC_HSI48_OFF

HSI48 clock deactivation

RCC_HSI48_ON

HSI48 clock activation

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSICALIBRATION_DEFAULT

Interrupts

RCC_IT_LSIRDY

LSI Ready Interrupt flag

RCC_IT_LSERDY

LSE Ready Interrupt flag

RCC_IT_HSIRDY

HSI16 Ready Interrupt flag

RCC_IT_HSERDY

HSE Ready Interrupt flag

RCC_IT_PLLRDY

PLL Ready Interrupt flag

RCC_IT_CSS

Clock Security System Interrupt flag

RCC_IT_LSECSS

LSE Clock Security System Interrupt flag

RCC_IT_HSI48RDY

HSI48 Ready Interrupt flag

LSE Drive Config

RCC_LSEDRIVE_LOW

LSE low drive capability

RCC_LSEDRIVE_MEDIUMLOW

LSE medium low drive capability

RCC_LSEDRIVE_MEDIUMHIGH

LSE medium high drive capability

RCC_LSEDRIVE_HIGH

LSE high drive capability

LSE Config

RCC_LSE_OFF

LSE clock deactivation

RCC_LSE_ON

LSE clock activation

RCC_LSE_BYPASS

External clock source for LSE clock

LSI Config

RCC_LSI_OFF

LSI clock deactivation

RCC_LSI_ON

LSI clock activation

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

MCO1 output disabled, no clock on MCO1

RCC_MCO1SOURCE_SYSCLK

SYSCLK selection as MCO1 source

RCC_MCO1SOURCE_HSI

HSI selection as MCO1 source

RCC_MCO1SOURCE_HSE

HSE selection as MCO1 source

RCC_MCO1SOURCE_PLLCLK

PLLCLK selection as MCO1 source

RCC_MCO1SOURCE_LSI

LSI selection as MCO1 source

RCC_MCO1SOURCE_LSE

LSE selection as MCO1 source

RCC_MCO1SOURCE_HSI48

HSI48 selection as MCO1 source

MCO1 Clock Prescaler

RCC_MCODIV_1

MCO not divided

RCC_MCODIV_2

MCO divided by 2

RCC_MCODIV_4

MCO divided by 4

RCC_MCODIV_8

MCO divided by 8

RCC_MCODIV_16

MCO divided by 16

MCO Index

RCC_MCO1

RCC_MCO

MCO1 to be compliant with other families with 2 MCOs

Oscillator Type

RCC_OSCILLATORTYPE_NONE

Oscillator configuration unchanged

RCC_OSCILLATORTYPE_HSE

HSE to configure

RCC_OSCILLATORTYPE_HSI

HSI to configure

RCC_OSCILLATORTYPE_LSE

LSE to configure

RCC_OSCILLATORTYPE_LSI

LSI to configure

RCC_OSCILLATORTYPE_HSI48

HSI48 to configure

PLLM Clock Divider

RCC_PLLM_DIV1

PLLM division factor = 1

RCC_PLLM_DIV2

PLLM division factor = 2

RCC_PLLM_DIV3

PLLM division factor = 3

RCC_PLLM_DIV4

PLLM division factor = 4

RCC_PLLM_DIV5

PLLM division factor = 5

RCC_PLLM_DIV6

PLLM division factor = 6

RCC_PLLM_DIV7

PLLM division factor = 7

RCC_PLLM_DIV8

PLLM division factor = 8

RCC_PLLM_DIV9

PLLM division factor = 9

RCC_PLLM_DIV10

PLLM division factor = 10

RCC_PLLM_DIV11

PLLM division factor = 11

RCC_PLLM_DIV12

PLLM division factor = 12

RCC_PLLM_DIV13

PLLM division factor = 13

RCC_PLLM_DIV14

PLLM division factor = 14

RCC_PLLM_DIV15

PLLM division factor = 15

RCC_PLLM_DIV16

PLLM division factor = 16

PLLP Clock Divider

RCC_PLLP_DIV2

PLLP division factor = 2

RCC_PLLP_DIV3

PLLP division factor = 3

RCC_PLLP_DIV4

PLLP division factor = 4

RCC_PLLP_DIV5

PLLP division factor = 5

RCC_PLLP_DIV6

PLLP division factor = 6

RCC_PLLP_DIV7

PLLP division factor = 7

RCC_PLLP_DIV8

PLLP division factor = 8

RCC_PLLP_DIV9

PLLP division factor = 9

RCC_PLLP_DIV10

PLLP division factor = 10

RCC_PLLP_DIV11

PLLP division factor = 11

RCC_PLLP_DIV12

PLLP division factor = 12

RCC_PLLP_DIV13

PLLP division factor = 13

RCC_PLLP_DIV14

PLLP division factor = 14

RCC_PLLP_DIV15

PLLP division factor = 15

RCC_PLLP_DIV16

PLLP division factor = 16

RCC_PLLP_DIV17

PLLP division factor = 17

RCC_PLLP_DIV18

PLLP division factor = 18

RCC_PLLP_DIV19

PLLP division factor = 19

RCC_PLLP_DIV20

PLLP division factor = 20

RCC_PLLP_DIV21

PLLP division factor = 21

RCC_PLLP_DIV22

PLLP division factor = 22

RCC_PLLP_DIV23

PLLP division factor = 23

RCC_PLLP_DIV24

PLLP division factor = 24

RCC_PLLP_DIV25

PLLP division factor = 25

RCC_PLLP_DIV26

PLLP division factor = 26

RCC_PLLP_DIV27

PLLP division factor = 27

RCC_PLLP_DIV28

PLLP division factor = 28

RCC_PLLP_DIV29

PLLP division factor = 29

RCC_PLLP_DIV30

PLLP division factor = 30

RCC_PLLP_DIV31

PLLP division factor = 31

PLLQ Clock Divider

RCC_PLLQ_DIV2

PLLQ division factor = 2

RCC_PLLQ_DIV4

PLLQ division factor = 4

RCC_PLLQ_DIV6

PLLQ division factor = 6

RCC_PLLQ_DIV8

PLLQ division factor = 8

PLLR Clock Divider

RCC_PLLR_DIV2

PLLR division factor = 2

RCC_PLLR_DIV4

PLLR division factor = 4

RCC_PLLR_DIV6

PLLR division factor = 6

RCC_PLLR_DIV8

PLLR division factor = 8

PLL Clock Output

RCC_PLL_ADCCLK

PLLADCCLK selection from main PLL

RCC_PLL_48M1CLK

PLL48M1CLK selection from main PLL

RCC_PLL_SYSCLK

PLLCLK selection from main PLL

PLL Clock Source

RCC_PLLSOURCE_NONE

No clock selected as PLL entry clock source

RCC_PLLSOURCE_HSI

HSI clock selected as PLL entry clock source

RCC_PLLSOURCE_HSE

HSE clock selected as PLL entry clock source

PLL Config

RCC_PLL_NONE

PLL configuration unchanged

RCC_PLL_OFF

PLL deactivation

RCC_PLL_ON

PLL activation

RCC RTC Clock Configuration

__HAL_RCC_RTC_ENABLE

Description:

- Macros to enable or disable the RTC clock.

Return value:

- None

Notes:

- As the RTC is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the RTC (to be done once after reset). These macros must be used after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

RTC Clock Source

RCC_RTCCLKSOURCE_NONE

No clock used as RTC clock

RCC_RTCCLKSOURCE_LSE

LSE oscillator clock used as RTC clock

RCC_RTCCLKSOURCE_LSI

LSI oscillator clock used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV32

HSE oscillator clock divided by 32 used as RTC clock

System Clock Source

RCC_SYSCLKSOURCE_HSI

HSI selection as system clock

RCC_SYSCLKSOURCE_HSE

HSE selection as system clock

RCC_SYSCLKSOURCE_PLLCLK

PLL selection as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI

HSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSE

HSE used as system clock

RCC_SYSCLKSOURCE_STATUS_PLLCLK

PLL used as system clock

System Clock Type

RCC_CLOCKTYPE_SYSCLK

SYSCLK to configure

RCC_CLOCKTYPE_HCLK

HCLK to configure

RCC_CLOCKTYPE_PCLK1

PCLK1 to configure

RCC_CLOCKTYPE_PCLK2

PCLK2 to configure

Timeout Values

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

46 HAL RCC Extension Driver

46.1 RCCEx Firmware driver registers structures

46.1.1 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32g4xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t Usart2ClockSelection***
- ***uint32_t Usart3ClockSelection***
- ***uint32_t Uart4ClockSelection***
- ***uint32_t Uart5ClockSelection***
- ***uint32_t Lpuart1ClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2c2ClockSelection***
- ***uint32_t I2c3ClockSelection***
- ***uint32_t I2c4ClockSelection***
- ***uint32_t Lptim1ClockSelection***
- ***uint32_t Sai1ClockSelection***
- ***uint32_t I2sClockSelection***
- ***uint32_t FdcanClockSelection***
- ***uint32_t UsbClockSelection***
- ***uint32_t RngClockSelection***
- ***uint32_t Adc12ClockSelection***
- ***uint32_t Adc345ClockSelection***
- ***uint32_t QspiClockSelection***
- ***uint32_t RTCClockSelection***

Field Documentation

- ***uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  Specifies USART1 clock source. This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
  Specifies USART2 clock source. This parameter can be a value of ***RCCEx_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection***
  Specifies USART3 clock source. This parameter can be a value of ***RCCEx_USART3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart4ClockSelection***
  Specifies UART4 clock source. This parameter can be a value of ***RCCEx_UART4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart5ClockSelection***
  Specifies UART5 clock source. This parameter can be a value of ***RCCEx_UART5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
  Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
  Specifies I2C1 clock source. This parameter can be a value of ***RCCEx_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection***
  Specifies I2C2 clock source. This parameter can be a value of ***RCCEx_I2C2_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
  Specifies I2C3 clock source. This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection***
  Specifies I2C4 clock source. This parameter can be a value of ***RCCEx_I2C4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection***
  Specifies LPTIM1 clock source. This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection***
  Specifies SAI1 clock source. This parameter can be a value of ***RCCEx_SAI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2sClockSelection***
  Specifies I2S clock source. This parameter can be a value of ***RCCEx_I2S_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::FdcanClockSelection***
  Specifies FDCAN clock source. This parameter can be a value of ***RCCEx_FDCAN_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection***
  Specifies USB clock source (warning: same source for RNG). This parameter can be a value of ***RCCEx_USB_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RngClockSelection***
  Specifies RNG clock source (warning: same source for USB). This parameter can be a value of ***RCCEx_RNG_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Adc12ClockSelection***
  Specifies ADC12 interface clock source. This parameter can be a value of ***RCCEx_ADC12_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Adc345ClockSelection***
  Specifies ADC345 interface clock source. This parameter can be a value of ***RCCEx_ADC345_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::QspiClockSelection***
  Specifies QuadSPI clock source. This parameter can be a value of ***RCCEx_QSPI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***

46.1.2 RCC_CRSInitTypeDef

RCC_CRSInitTypeDef is defined in the stm32g4xx_hal_rcc_ex.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t ReloadValue***
- ***uint32_t ErrorLimitValue***
- ***uint32_t HSI48CalibrationValue***

Field Documentation

- ***uint32_t RCC_CRSInitTypeDef::Prescaler***
  Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEx_CRS_SynchroDivider***
- ***uint32_t RCC_CRSInitTypeDef::Source***
  Specifies the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroSource***
- ***uint32_t RCC_CRSInitTypeDef::Polarity***
  Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroPolarity***
- ***uint32_t RCC_CRSInitTypeDef::ReloadValue***
  Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro **__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEx_CRS_ReloadValueDefault*** .

- ***uint32_t RCC_CRSInitTypeDef::ErrorLimitValue***

  Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEx_CRS_ErrorLimitDefault***
- ***uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue***

  Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x7F or a value of ***RCCEx_CRS_HSI48CalibrationDefault***

46.1.3 RCC_CRSSynchroInfoTypeDef

RCC_CRSSynchroInfoTypeDef is defined in the stm32g4xx_hal_rcc_ex.h

Data Fields

- ***uint32_t ReloadValue***
- ***uint32_t HSI48CalibrationValue***
- ***uint32_t FreqErrorCapture***
- ***uint32_t FreqErrorDirection***

Field Documentation

- ***uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue***

  Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue***

  Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x7F
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture***

  Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection***

  Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEx_CRS_FreqErrorDirection***

46.2 RCCEx Firmware driver API description

The following section lists the various functions of the RCCEx library.

46.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Note: *Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.*

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig***
- ***HAL_RCCEx_GetPeriphCLKConfig***
- ***HAL_RCCEx_GetPeriphCLKFreq***

46.2.2 Extended clock management functions

This subsection provides a set of functions allowing to control the activation or deactivation of LSE CSS, Low speed clock output and clock after wake-up from STOP mode.

This section contains the following APIs:

- ***HAL_RCCEx_EnableLSECSS***
- ***HAL_RCCEx_DisableLSECSS***
- ***HAL_RCCEx_EnableLSECSS_IT***
- ***HAL_RCCEx_LSECSS_IRQHandler***
- ***HAL_RCCEx_LSECSS_Callback***

- *HAL_RCCEx_EnableLSCO*
- *HAL_RCCEx_DisableLSCO*

46.2.3 Extended Clock Recovery System Control functions

For devices with Clock Recovery System feature (CRS), RCC Extention HAL driver can be used as follows:

1. In System clock config, HSI48 needs to be enabled
2. Enable CRS clock in IP MSP init which will use CRS functions
3. Call CRS functions as follows:
   a. Prepare synchronization configuration necessary for HSI48 calibration
      - Default values can be set for frequency Error Measurement (reload and error limit) and also HSI48 oscillator smooth trimming.
      - Macro __HAL_RCC_CRS_RELOADVALUE_CALCULATE can be also used to calculate directly reload value with target and sychronization frequencies values

   b. Call function HAL_RCCEx_CRSConfig which
      - Resets CRS registers to their default values.
      - Configures CRS registers with synchronization configuration
      - Enables automatic calibration and frequency error counter feature Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF will not be generated by the host. No SYNC signal will therefore be provided to the CRS to calibrate the HSI48 on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.

   c. A polling function is provided to wait for complete synchronization
      - Call function HAL_RCCEx_CRSWaitSynchronization()
      - According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK
4. User can retrieve information related to synchronization in calling function HAL_RCCEx_CRSGetSynchronizationInfo()
5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again HAL_RCCEx_CRSConfig. Note: When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).
6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (CRS_IRQn/CRS_IRQHandler)
   – Call function HAL_RCCEx_CRSConfig()
   – Enable CRS_IRQn (thanks to NVIC functions)
   – Enable CRS interrupt (__HAL_RCC_CRS_ENABLE_IT)
   – Implement CRS status management in the following user callbacks called from HAL_RCCEx_CRS_IRQHandler():
      - HAL_RCCEx_CRS_SyncOkCallback()
      - HAL_RCCEx_CRS_SyncWarnCallback()
      - HAL_RCCEx_CRS_ExpectedSyncCallback()
      - HAL_RCCEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function HAL_RCCEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling HAL_RCCEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:

- *HAL_RCCEx_CRSConfig*
- *HAL_RCCEx_CRSSoftwareSynchronizationGenerate*
- *HAL_RCCEx_CRSGetSynchronizationInfo*
- *HAL_RCCEx_CRSWaitSynchronization*
- *HAL_RCCEx_CRS_IRQHandler*

- *HAL_RCCEx_CRS_SyncOkCallback*
- *HAL_RCCEx_CRS_SyncWarnCallback*
- *HAL_RCCEx_CRS_ExpectedSyncCallback*
- *HAL_RCCEx_CRS_ErrorCallback*

46.2.4 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Initialize the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains a field PeriphClockSelection which can be a combination of the following values:
  - RCC_PERIPHCLK_RTC RTC peripheral clock
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USART2 USART2 peripheral clock
  - RCC_PERIPHCLK_USART3 USART3 peripheral clock
  - RCC_PERIPHCLK_UART4 UART4 peripheral clock (only for devices with UART4)
  - RCC_PERIPHCLK_UART5 UART5 peripheral clock (only for devices with UART5)
  - RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2C2 I2C2 peripheral clock
  - RCC_PERIPHCLK_I2C3 I2C3 peripheral clock
  - RCC_PERIPHCLK_I2C4 I2C4 peripheral clock (only for devices with I2C4)
  - RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock
  - RCC_PERIPHCLK_SAI1 SAI1 peripheral clock
  - RCC_PERIPHCLK_I2S I2S peripheral clock
  - RCC_PERIPHCLK_FDCAN FDCAN peripheral clock (only for devices with FDCAN)
  - RCC_PERIPHCLK_RNG RNG peripheral clock
  - RCC_PERIPHCLK_USB USB peripheral clock (only for devices with USB)
  - RCC_PERIPHCLK_ADC12 ADC1 and ADC2 peripheral clock
  - RCC_PERIPHCLK_ADC345 ADC3, ADC4 and ADC5 peripheral clock (only for devices with ADC3, ADC4, ADC5)
  - RCC_PERIPHCLK_QSPI QuadSPI peripheral clock (only for devices with QuadSPI)

Return values

- **HAL:** status

Notes

- Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source: in this case the access to Backup domain is enabled.

HAL_RCCEx_GetPeriphCLKConfig

Function name

void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(USART1, USART2, USART3, UART4, UART5, LPUART1, I2C1, I2C2, I2C3, I2C4, LPTIM1, SAI1, I2Sx, FDCANx, USB, RNG, ADCx, RTC, QSPI).

Return values

- **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name

uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)

Function description

Return the peripheral clock frequency for peripherals with clock source from PLL.

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USART2 USART2 peripheral clock
  - RCC_PERIPHCLK_USART3 USART3 peripheral clock
  - RCC_PERIPHCLK_UART4 UART4 peripheral clock (only for devices with UART4)
  - RCC_PERIPHCLK_UART5 UART5 peripheral clock (only for devices with UART5)
  - RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2C2 I2C2 peripheral clock
  - RCC_PERIPHCLK_I2C3 I2C3 peripheral clock
  - RCC_PERIPHCLK_I2C4 I2C4 peripheral clock (only for devices with I2C4)
  - RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock
  - RCC_PERIPHCLK_SAI1 SAI1 peripheral clock
  - RCC_PERIPHCLK_I2S SPI peripheral clock
  - RCC_PERIPHCLK_FDCAN FDCAN peripheral clock (only for devices with FDCAN)
  - RCC_PERIPHCLK_RNG RNG peripheral clock
  - RCC_PERIPHCLK_USB USB peripheral clock (only for devices with USB)
  - RCC_PERIPHCLK_ADC12 ADC1 and ADC2 peripheral clock
  - RCC_PERIPHCLK_ADC345 ADC3, ADC4 and ADC5 peripheral clock (only for devices with ADC3, ADC4, ADC5)
  - RCC_PERIPHCLK_QSPI QSPI peripheral clock (only for devices with QSPI)
  - RCC_PERIPHCLK_RTC RTC peripheral clock

Return values

- **Frequency:** in Hz

Notes

- Return 0 if peripheral clock identifier not managed by this API

HAL_RCCEx_EnableLSECSS

Function name

void HAL_RCCEx_EnableLSECSS (void)

Function description

Enable the LSE Clock Security System.

Return values

- **None:**

Notes

- Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEx_PeriphCLKConfig().

HAL_RCCEx_DisableLSECSS

Function name

void HAL_RCCEx_DisableLSECSS (void)

Function description

Disable the LSE Clock Security System.

Return values

- **None:**

Notes

- LSE Clock Security System can only be disabled after a LSE failure detection.

HAL_RCCEx_EnableLSECSS_IT

Function name

void HAL_RCCEx_EnableLSECSS_IT (void)

Function description

Enable the LSE Clock Security System Interrupt & corresponding EXTI line.

Return values

- **None:**

Notes

- LSE Clock Security System Interrupt is mapped on RTC EXTI line 19

HAL_RCCEx_LSECSS_IRQHandler

Function name

void HAL_RCCEx_LSECSS_IRQHandler (void)

Function description

Handle the RCC LSE Clock Security System interrupt request.

Return values

- **None:**

HAL_RCCEx_LSECSS_Callback

Function name

void HAL_RCCEx_LSECSS_Callback (void)

Function description

RCCEx LSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCCEx_EnableLSCO

Function name

void HAL_RCCEx_EnableLSCO (uint32_t LSCOSource)

Function description

Select the Low Speed clock source to output on LSCO pin (PA2).

Parameters

- **LSCOSource:** specifies the Low Speed clock source to output. This parameter can be one of the following values:
  - RCC_LSCOSOURCE_LSI LSI clock selected as LSCO source
  - RCC_LSCOSOURCE_LSE LSE clock selected as LSCO source

Return values

- **None:**

HAL_RCCEx_DisableLSCO

Function name

void HAL_RCCEx_DisableLSCO (void)

Function description

Disable the Low Speed clock output.

Return values

- **None:**

HAL_RCCEx_CRSConfig

Function name

void HAL_RCCEx_CRSConfig (RCC_CRSInitTypeDef * pInit)

Function description

Start automatic synchronization for polling mode.

Parameters

- **pInit:** Pointer on RCC_CRSInitTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function name

void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)

Function description

Generate the software synchronization event.

Return values

- **None:**

HAL_RCCEx_CRSGetSynchronizationInfo

Function name

void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)

Function description

Return synchronization info.

Parameters

- **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSWaitSynchronization

Function name

uint32_t HAL_RCCEx_CRSWaitSynchronization (uint32_t Timeout)

Function description

Wait for CRS Synchronization status.

Parameters

- **Timeout:** Duration of the timeout

Return values

- **Combination:** of Synchronization status This parameter can be a combination of the following values:
  - RCC_CRS_TIMEOUT
  - RCC_CRS_SYNCOK
  - RCC_CRS_SYNCWARN
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Notes

- Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
- If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEx_CRS_IRQHandler

Function name

void HAL_RCCEx_CRS_IRQHandler (void)

Function description

Handle the Clock Recovery System interrupt request.

Return values

- **None:**

HAL_RCCEx_CRS_SyncOkCallback

Function name

void HAL_RCCEx_CRS_SyncOkCallback (void)

Function description

RCCEx Clock Recovery System SYNCOK interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_SyncWarnCallback

Function name

void HAL_RCCEx_CRS_SyncWarnCallback (void)

Function description

RCCEx Clock Recovery System SYNCWARN interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ExpectedSyncCallback

Function name

void HAL_RCCEx_CRS_ExpectedSyncCallback (void)

Function description

RCCEx Clock Recovery System Expected SYNC interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ErrorCallback

Function name

void HAL_RCCEx_CRS_ErrorCallback (uint32_t Error)

Function description

RCCEx Clock Recovery System Error interrupt callback.

Parameters

- **Error:** Combination of Error status. This parameter can be a combination of the following values:
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Return values

- **none:**

46.3 RCCEx Firmware driver defines

The following section lists the various define and macros of the module.

46.3.1 RCCEx

RCCEx

ADC12 Clock Source

RCC_ADC12CLKSOURCE_NONE

RCC_ADC12CLKSOURCE_PLL

RCC_ADC12CLKSOURCE_SYSCLK

ADC345 Clock Source

RCC_ADC345CLKSOURCE_NONE

RCC_ADC345CLKSOURCE_PLL

RCC_ADC345CLKSOURCE_SYSCLK

RCCEx CRS ErrorLimitDefault

RCC_CRS_ERRORLIMIT_DEFAULT

Default Frequency error limit

RCCEx CRS Extended Features

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE

Description:

- Enable the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE

Description:

- Disable the oscillator clock for frequency error counter.

Return value:

- None

__HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enable the automatic hardware adjustement of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enable or disable the automatic hardware adjustement of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

RCCEx CRS Flags

RCC_CRS_FLAG_SYNCOK

SYNC event OK flag

RCC_CRS_FLAG_SYNCWARN

SYNC warning flag

RCC_CRS_FLAG_ERR

Error flag

RCC_CRS_FLAG_ESYNC

Expected SYNC flag

RCC_CRS_FLAG_SYNCERR

SYNC error

RCC_CRS_FLAG_SYNCMISS

SYNC missed

RCC_CRS_FLAG_TRIMOVF

Trimming overflow or underflow

RCCEx CRS FreqErrorDirection

RCC_CRS_FREQERRORDIR_UP

Upcounting direction, the actual frequency is above the target

RCC_CRS_FREQERRORDIR_DOWN

Downcounting direction, the actual frequency is below the target

RCCEx CRS HSI48CalibrationDefault

RCC_CRS_HSI48CALIBRATION_DEFAULT

The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK

SYNC event OK

RCC_CRS_IT_SYNCWARN

SYNC warning

RCC_CRS_IT_ERR

Error

RCC_CRS_IT_ESYNC

Expected SYNC

RCC_CRS_IT_SYNCERR

SYNC error

RCC_CRS_IT_SYNCMISS

SYNC missed

RCC_CRS_IT_TRIMOVF

Trimming overflow or underflow

RCCEx CRS ReloadValueDefault

RCC_CRS_RELOADVALUE_DEFAULT

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEx CRS Status

RCC_CRS_NONE

RCC_CRS_TIMEOUT

RCC_CRS_SYNCOK

RCC_CRS_SYNCWARN

RCC_CRS_SYNCERR

RCC_CRS_SYNCMISS

RCC_CRS_TRIMOVF

RCCEx CRS SynchroDivider

RCC_CRS_SYNC_DIV1

Synchro Signal not divided (default)

RCC_CRS_SYNC_DIV2

Synchro Signal divided by 2

RCC_CRS_SYNC_DIV4

Synchro Signal divided by 4

RCC_CRS_SYNC_DIV8

Synchro Signal divided by 8

RCC_CRS_SYNC_DIV16

Synchro Signal divided by 16

RCC_CRS_SYNC_DIV32

Synchro Signal divided by 32

RCC_CRS_SYNC_DIV64

Synchro Signal divided by 64

RCC_CRS_SYNC_DIV128

Synchro Signal divided by 128

RCCEx CRS SynchroPolarity

RCC_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

RCC_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

RCCEx CRS SynchroSource

RCC_CRS_SYNC_SOURCE_GPIO

Synchro Signal source GPIO

RCC_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

RCC_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

RCCEx Exported Macros

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1_CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

Return value:

- None

__HAL_RCC_GET_USART1_SOURCE

Description:

- Macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2_CLKSOURCE__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

Return value:

- None

__HAL_RCC_GET_USART2_SOURCE

Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- Macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3_CLKSOURCE__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

Return value:

- None

__HAL_RCC_GET_USART3_SOURCE

Description:

- Macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

__HAL_RCC_UART4_CONFIG

Description:

- Macro to configure the UART4 clock (UART4CLK).

Parameters:

- __UART4_CLKSOURCE__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK System Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

Return value:

- None

__HAL_RCC_GET_UART4_SOURCE

Description:

- Macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK System Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- Macro to configure the UART5 clock (UART5CLK).

Parameters:

- __UART5_CLKSOURCE__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

Return value:

- None

__HAL_RCC_GET_UART5_SOURCE

Description:

- Macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

__HAL_RCC_LPUART1_CONFIG

Description:

- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

Return value:

- None

__HAL_RCC_GET_LPUART1_SOURCE

Description:

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1_CLKSOURCE__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

Return value:

- None

__HAL_RCC_GET_I2C1_SOURCE

Description:

- Macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- Macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2_CLKSOURCE__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK System Clock selected as I2C2 clock

Return value:

- None

__HAL_RCC_GET_I2C2_SOURCE

Description:

- Macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK System Clock selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3_CLKSOURCE__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_SYSCLK System Clock selected as I2C3 clock

Return value:

- None

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK1 PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_SYSCLK System Clock selected as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- Macro to configure the I2C4 clock (I2C4CLK).

Parameters:

- __I2C4_CLKSOURCE__: specifies the I2C4 clock source. This parameter can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_SYSCLK System Clock selected as I2C4 clock

Return value:

- None

__HAL_RCC_GET_I2C4_SOURCE

Description:

- Macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_SYSCLK System Clock selected as I2C4 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

Parameters:

- __LPTIM1_CLKSOURCE__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK1 PCLK1 selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_HSI LSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock

Return value:

- None

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK1 PCLK1 selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_HSI System Clock selected as LPUART1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_SAI1_CONFIG

Description:

- Macro to configure the SAI1 clock source.

Parameters:

- __SAI1_CLKSOURCE__: defines the SAI1 clock source. This clock is derived from the HSI, system PLL, System Clock or external clock. This parameter can be one of the following values:
  - RCC_SAI1CLKSOURCE_SYSCLK SAI1 clock = System Clock
  - RCC_SAI1CLKSOURCE_PLL SAI1 clock = PLL "Q" clock
  - RCC_SAI1CLKSOURCE_EXT SAI1 clock = EXT
  - RCC_SAI1CLKSOURCE_HSI SAI1 clock = HSI

Return value:

- None

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI1CLKSOURCE_SYSCLK SAI1 clock = System Clock
  - RCC_SAI1CLKSOURCE_PLL SAI1 clock = PLL "Q" clock
  - RCC_SAI1CLKSOURCE_EXT SAI1 clock = EXT
  - RCC_SAI1CLKSOURCE_HSI SAI1 clock = HSI

__HAL_RCC_I2S_CONFIG

Description:

- Macro to configure the I2S clock source.

Parameters:

- __I2S_CLKSOURCE__: defines the I2S clock source. This clock is derived from the HSI, system PLL, System Clock or external clock. This parameter can be one of the following values:
  - RCC_I2SCLKSOURCE_SYSCLK I2S clock = System Clock
  - RCC_I2SCLKSOURCE_PLL I2S clock = PLL "Q" clock
  - RCC_I2SCLKSOURCE_EXT I2S clock = EXT
  - RCC_I2SCLKSOURCE_HSI I2S clock = HSI

Return value:

- None

__HAL_RCC_GET_I2S_SOURCE

Description:

- Macro to get the I2S clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2SCLKSOURCE_SYSCLK I2S clock = System Clock
  - RCC_I2SCLKSOURCE_PLL I2S clock = PLL "Q" clock
  - RCC_I2SCLKSOURCE_EXT I2S clock = EXT
  - RCC_I2SCLKSOURCE_HSI I2S clock = HSI

__HAL_RCC_FDCAN_CONFIG

Description:

- Macro to configure the FDCAN clock source.

Parameters:

- __FDCAN_CLKSOURCE__: defines the FDCAN clock source. This clock is derived from the HSE, system PLL or PCLK1. This parameter can be one of the following values:
  - RCC_FDCANCLKSOURCE_HSE FDCAN clock = HSE
  - RCC_FDCANCLKSOURCE_PLL FDCAN clock = PLL "Q" clock
  - RCC_FDCANCLKSOURCE_PCLK1 FDCAN clock = PCLK1

Return value:

- None

__HAL_RCC_GET_FDCAN_SOURCE

Description:

- Macro to get the FDCAN clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_FDCANCLKSOURCE_HSE FDCAN clock = HSE
  - RCC_FDCANCLKSOURCE_PLL FDCAN clock = PLL "Q" clock
  - RCC_FDCANCLKSOURCE_PCLK1 FDCAN clock = PCLK1

__HAL_RCC_RNG_CONFIG

Description:

- Macro to configure the RNG clock.

Parameters:

- __RNG_CLKSOURCE__: specifies the RNG clock source. This parameter can be one of the following values:
  - RCC_RNGCLKSOURCE_HSI48 HSI48 selected as RNG clock for devices with HSI48
  - RCC_RNGCLKSOURCE_PLL PLL Clock selected as RNG clock

Return value:

- None

Notes:

- USB and RNG peripherals share the same 48MHz clock source.

__HAL_RCC_GET_RNG_SOURCE

Description:

- Macro to get the RNG clock.

Return value:

- The: clock source can be one of the following values:
  - RCC_RNGCLKSOURCE_HSI48 HSI48 selected as RNG clock for devices with HSI48
  - RCC_RNGCLKSOURCE_PLL PLL "Q" clock selected as RNG clock

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USB_CLKSOURCE__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBCLKSOURCE_HSI48 HSI48 selected as 48MHz clock for devices with HSI48
  - RCC_USBCLKSOURCE_PLL PLL "Q" clock (PLL48M1CLK) selected as USB clock

Return value:

- None

Notes:

- USB, RNG peripherals share the same 48MHz clock source.

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBCLKSOURCE_HSI48 HSI48 selected as 48MHz clock for devices with HSI48
  - RCC_USBCLKSOURCE_PLL PLL "Q" clock (PLL48M1CLK) selected as USB clock

__HAL_RCC_ADC12_CONFIG

Description:

- Macro to configure the ADC12 interface clock.

Parameters:

- __ADC12_CLKSOURCE__: specifies the ADC12 digital interface clock source. This parameter can be one of the following values:
  - RCC_ADC12CLKSOURCE_NONE No clock selected as ADC12 clock
  - RCC_ADC12CLKSOURCE_PLL PLL Clock selected as ADC12 clock
  - RCC_ADC12CLKSOURCE_SYSCLK System Clock selected as ADC12 clock

Return value:

- None

__HAL_RCC_GET_ADC12_SOURCE

Description:

- Macro to get the ADC12 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADC12CLKSOURCE_NONE No clock selected as ADC12 clock
  - RCC_ADC12CLKSOURCE_PLL PLL Clock selected as ADC12 clock
  - RCC_ADC12CLKSOURCE_SYSCLK System Clock selected as ADC12 clock

__HAL_RCC_ADC345_CONFIG

Description:

- Macro to configure the ADC345 interface clock.

Parameters:

- __ADC345_CLKSOURCE__: specifies the ADC345 digital interface clock source. This parameter can be one of the following values:
  - RCC_ADC345CLKSOURCE_NONE No clock selected as ADC345 clock
  - RCC_ADC345CLKSOURCE_PLL PLL Clock selected as ADC345 clock
  - RCC_ADC345CLKSOURCE_SYSCLK System Clock selected as ADC345 clock

Return value:

- None

__HAL_RCC_GET_ADC345_SOURCE

Description:

- Macro to get the ADC345 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADC345CLKSOURCE_NONE No clock selected as ADC345 clock
  - RCC_ADC345CLKSOURCE_PLL PLL Clock selected as ADC345 clock
  - RCC_ADC345CLKSOURCE_SYSCLK System Clock selected as ADC345 clock

__HAL_RCC_QSPI_CONFIG

Description:

- Macro to configure the QuadSPI clock.

Parameters:

- __QSPI_CLKSOURCE__: specifies the QuadSPI clock source. This parameter can be one of the following values:
  - RCC_QSPICLKSOURCE_SYSCLK System Clock selected as QuadSPI clock
  - RCC_QSPICLKSOURCE_HSI HSI clock selected as QuadSPI clock
  - RCC_QSPICLKSOURCE_PLL PLL Q divider clock selected as QuadSPI clock

Return value:

- None

__HAL_RCC_GET_QSPI_SOURCE

Description:

- Macro to get the QuadSPI clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_QSPICLKSOURCE_SYSCLK System Clock selected as QuadSPI clock
  - RCC_QSPICLKSOURCE_HSI HSI clock selected as QuadSPI clock
  - RCC_QSPICLKSOURCE_PLL PLL Q divider clock selected as QuadSPI clock

RCC LSE CSS external interrupt line

RCC_EXTI_LINE_LSECSS

External interrupt line 19 connected to the LSE CSS EXTI Line

FDCAN Clock Source

RCC_FDCANCLKSOURCE_HSE

RCC_FDCANCLKSOURCE_PLL

RCC_FDCANCLKSOURCE_PCLK1

Flags Interrupts Management

__HAL_RCC_LSECSS_EXTI_ENABLE_IT

Description:

- Enable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_DISABLE_IT

Description:

- Disable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT

Description:

- Enable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT

Description:

- Disable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the RCC LSE CSS Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the RCC LSE CSS Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_GET_FLAG

Description:

- Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not.

Return value:

- EXTI: RCC LSE CSS Line Status.

__HAL_RCC_LSECSS_EXTI_CLEAR_FLAG

Description:

- Clear the RCC LSE CSS EXTI flag.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RCC LSE CSS EXTI line.

Return value:

- None.

__HAL_RCC_CRS_ENABLE_IT

Description:

- Enable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_DISABLE_IT

Description:

- Disable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_GET_IT_SOURCE

Description:

- Check whether the CRS interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt source to check. This parameter can be one of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

RCC_CRS_IT_ERROR_MASK

Description:

- Clear the CRS interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt
  - RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
  - RCC_CRS_IT_SYNCERR SYNC error interrupt
  - RCC_CRS_IT_SYNCMISS SYNC missed interrupt

__HAL_RCC_CRS_CLEAR_IT

__HAL_RCC_CRS_GET_FLAG

Description:

- Check whether the specified CRS flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- The: new state of _FLAG_ (TRUE or FALSE).

RCC_CRS_FLAG_ERROR_MASK

Description:

- Clear the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

__HAL_RCC_CRS_CLEAR_FLAG

I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

RCC_I2C1CLKSOURCE_SYSCLK

RCC_I2C1CLKSOURCE_HSI

I2C2 Clock Source

RCC_I2C2CLKSOURCE_PCLK1

RCC_I2C2CLKSOURCE_SYSCLK

RCC_I2C2CLKSOURCE_HSI

I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK1

RCC_I2C3CLKSOURCE_SYSCLK

RCC_I2C3CLKSOURCE_HSI

I2C4 Clock Source

RCC_I2C4CLKSOURCE_PCLK1

RCC_I2C4CLKSOURCE_SYSCLK

RCC_I2C4CLKSOURCE_HSI

I2S Clock Source

RCC_I2SCLKSOURCE_SYSCLK

RCC_I2SCLKSOURCE_PLL

RCC_I2SCLKSOURCE_EXT

RCC_I2SCLKSOURCE_HSI

LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_PCLK1

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_HSI

RCC_LPTIM1CLKSOURCE_LSE

LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_PCLK1

RCC_LPUART1CLKSOURCE_SYSCLK

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_LSE

Low Speed Clock Source

RCC_LSCOSOURCE_LSI

LSI selection for low speed clock output

RCC_LSCOSOURCE_LSE

LSE selection for low speed clock output

Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_SAI1

RCC_PERIPHCLK_I2S

RCC_PERIPHCLK_FDCAN

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_RNG

RCC_PERIPHCLK_ADC12

RCC_PERIPHCLK_ADC345

RCC_PERIPHCLK_I2C4

RCC_PERIPHCLK_QSPI

RCC_PERIPHCLK_RTC

QuadSPI Clock Source

RCC_QSPICLKSOURCE_SYSCLK

RCC_QSPICLKSOURCE_HSI

RCC_QSPICLKSOURCE_PLL

RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLL

SAI1 Clock Source

RCC_SAI1CLKSOURCE_SYSCLK

RCC_SAI1CLKSOURCE_PLL

RCC_SAI1CLKSOURCE_EXT

RCC_SAI1CLKSOURCE_HSI

UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1

RCC_UART4CLKSOURCE_SYSCLK

RCC_UART4CLKSOURCE_HSI

RCC_UART4CLKSOURCE_LSE

UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1

RCC_UART5CLKSOURCE_SYSCLK

RCC_UART5CLKSOURCE_HSI

RCC_UART5CLKSOURCE_LSE

USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_SYSCLK

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_LSE

USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_SYSCLK

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_LSE

USART3 Clock Source

RCC_USART3CLKSOURCE_PCLK1

RCC_USART3CLKSOURCE_SYSCLK

RCC_USART3CLKSOURCE_HSI

RCC_USART3CLKSOURCE_LSE

USB Clock Source

RCC_USBCLKSOURCE_HSI48

RCC_USBCLKSOURCE_PLL

47 HAL RNG Generic Driver

47.1 RNG Firmware driver registers structures

47.1.1 RNG_InitTypeDef

RNG_InitTypeDef is defined in the stm32g4xx_hal_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  CED Clock error detection

47.1.2 RNG_HandleTypeDef

RNG_HandleTypeDef is defined in the stm32g4xx_hal_rng.h

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t RandomNumber***

Field Documentation

- ***RNG_TypeDef* RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef RNG_HandleTypeDef::State***
  RNG communication state
- ***__IO uint32_t RNG_HandleTypeDef::ErrorCode***
  RNG Error code
- ***uint32_t RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data

47.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

47.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL_RNG_GenerateRandomNumber() function.

47.2.2 Callback registration

The compilation define USE_HAL_RNG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_RNG_RegisterCallback() to register a user callback. Function @ref HAL_RNG_RegisterCallback() allows to register following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_RNG_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_RNG_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit.

For specific callback ReadyDataCallback, use dedicated register callbacks: respectively @ref HAL_RNG_RegisterReadyDataCallback() , @ref HAL_RNG_UnRegisterReadyDataCallback().

By default, after the @ref HAL_RNG_Init() and when the state is HAL_RNG_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: example @ref HAL_RNG_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_RNG_Init() and @ref HAL_RNG_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_RNG_Init() and @ref HAL_RNG_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_RNG_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_RNG_STATE_READY or HAL_RNG_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_RNG_RegisterCallback() before calling @ref HAL_RNG_DeInit() or @ref HAL_RNG_Init() function.

When The compilation define USE_HAL_RNG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

47.2.3 Initialization and configuration functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- ***HAL_RNG_Init***
- ***HAL_RNG_DeInit***
- ***HAL_RNG_MspInit***
- ***HAL_RNG_MspDeInit***

47.2.4 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- ***HAL_RNG_GenerateRandomNumber***
- ***HAL_RNG_GenerateRandomNumber_IT***
- ***HAL_RNG_IRQHandler***

- *HAL_RNG_ReadLastRandomNumber*
- *HAL_RNG_ReadyDataCallback*
- *HAL_RNG_ErrorCallback*

47.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.
This section contains the following APIs:

- *HAL_RNG_GetState*
- *HAL_RNG_GetError*

47.2.6 Detailed description of functions

HAL_RNG_Init

Function name

HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG peripheral and creates the associated handle.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_DeInit

Function name

HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG peripheral.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_MspInit

Function name

void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_MspDeInit

Function name

void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_GenerateRandomNumber

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)

Function description

Generates a 32-bit random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** pointer to generated random number variable if successful.

Return values

- **HAL:** status

Notes

- This function checks value of RNG_FLAG_DRDY flag to know if valid random number is available in the DR register (RNG_FLAG_DRDY flag set whenever a random number is available through the RNG_DR register). After transitioning from 0 to 1 (random number available), RNG_FLAG_DRDY flag remains high until output buffer becomes empty after reading four words from the RNG_DR register, i.e. further function calls will immediately return a new u32 random number (additional words are available and can be read by the application, till RNG_FLAG_DRDY flag remains high).
- When no more random number data is available in DR register, RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)

Function description

Generates a 32-bit random number in interrupt mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)

Function description

Read latest generated random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **random:** value

HAL_RNG_IRQHandler

Function name

void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)

Function description

Handles RNG interrupt request.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.
- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name

void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)

Function description

RNG error callbacks.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_ReadyDataCallback

Function name

void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)

Function description

Data Ready callback in non-blocking mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** generated random number.

Return values

- **None:**

Notes

- When RNG_FLAG_DRDY flag value is set, first random number has been read from DR register in IRQ Handler and is provided as callback parameter. Depending on valid data available in the conditioning output buffer, additional words can be read by the application from DR register till DRDY bit remains high.

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)

Function description

Returns the RNG state.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** state

HAL_RNG_GetError

Function name

uint32_t HAL_RNG_GetError (RNG_HandleTypeDef * hrng)

Function description

Return the RNG handle error code.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **RNG:** Error Code

47.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

47.3.1 RNG

RNG

RNG Error Definition

HAL_RNG_ERROR_NONE

No error

HAL_RNG_ERROR_TIMEOUT

Timeout error

HAL_RNG_ERROR_BUSY

Busy error

HAL_RNG_ERROR_SEED

Seed error

HAL_RNG_ERROR_CLOCK

Clock error

RNG Interrupt definition

RNG_IT_DRDY

Data Ready interrupt

RNG_IT_CEI

Clock error interrupt

RNG_IT_SEI

Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY

Data ready

RNG_FLAG_CECS

Clock error current status

RNG_FLAG_SECS

Seed error current status

RNG Clock Error Detection

RNG_CED_ENABLE

Clock error detection Enabled

RNG_CED_DISABLE

Clock error detection Disabled

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG

Description:

- Check the selected RNG flag status.

Parameters:

- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  - RNG_FLAG_DRDY: Data ready
  - RNG_FLAG_CECS: Clock error current status
  - RNG_FLAG_SECS: Seed error current status

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_RNG_CLEAR_FLAG

Description:

- Clears the selected RNG flag status.

Parameters:

- __HANDLE__: RNG handle
- __FLAG__: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.

__HAL_RNG_ENABLE_IT

Description:

- Enables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE_IT

Description:

- Disables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_IT

Description:

- Checks whether the specified RNG interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RNG_CLEAR_IT

Description:

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

48 HAL RTC Generic Driver

48.1 RTC Firmware driver registers structures

48.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32g4xx_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***
- ***uint32_t OutPutPullUp***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***

  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***

  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***

  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***

  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTCEx_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutRemap***

  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***

  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***

  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::OutPutPullUp***

  Specifies the RTC Output Pull-Up mode. This parameter can be a value of ***RTC_Output_PullUp_ALARM_OUT***

48.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32g4xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***
- ***uint8_t TimeFormat***
- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***uint32_t DayLightSaving***

- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***

  Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***

  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***

  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***

  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::SubSeconds***

  Specifies the RTC_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- ***uint32_t RTC_TimeTypeDef::SecondFraction***

  Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***

  Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of ***RTC_DayLightSaving_Definitions***
- ***uint32_t RTC_TimeTypeDef::StoreOperation***

  Specifies RTC_StoreOperation value to be written in the BKP bit in CR register to store the operation. This parameter can be a value of ***RTC_StoreOperation_Definitions***

48.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32g4xx_hal_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Date***
- ***uint8_t Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***

  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::Month***

  Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::Date***

  Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***

  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

48.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32g4xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***

- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of ***RTC_Alarm_Sub_Seconds_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm . This parameter can be a value of ***RTC_Alarms_Definitions***

48.1.5 RTC_HandleTypeDef

RTC_HandleTypeDef is defined in the stm32g4xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***

Field Documentation

- ***RTC_TypeDef* RTC_HandleTypeDef::Instance***
  Legacy register base address. Not used anymore, the driver directly uses cmsis base address
- ***RTC_InitTypeDef RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State***
  Time communication state

48.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

48.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

48.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).

2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

48.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the __HAL_RCC_PWR_CLK_ENABLE() function.
- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.
- Select the RTC clock source using the __HAL_RCC_RTC_CONFIG() function.
- Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() function.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSEdiv32)
2. Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() macro.

48.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

48.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

Callback registration

The compilation define USE_RTC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Function @ref HAL_RTC_RegisterCallback() to register an interrupt callback.

Function @ref HAL_RTC_RegisterCallback() allows to register following callbacks:

- AlarmAEventCallback : RTC Alarm A Event callback.
- AlarmBEventCallback : RTC Alarm B Event callback.
- TimeStampEventCallback : RTC TimeStamp Event callback.
- WakeUpTimerEventCallback : RTC WakeUpTimer Event callback.
- Tamper1EventCallback : RTC Tamper 1 Event callback.

- Tamper2EventCallback : RTC Tamper 2 Event callback.
- Tamper3EventCallback : RTC Tamper 3 Event callback.
- Tamper4EventCallback : RTC Tamper 4 Event callback.
- Tamper5EventCallback : RTC Tamper 5 Event callback.
- Tamper6EventCallback : RTC Tamper 6 Event callback.
- Tamper7EventCallback : RTC Tamper 7 Event callback.
- Tamper8EventCallback : RTC Tamper 8 Event callback.
- InternalTamper1EventCallback : RTC InternalTamper 1 Event callback.
- InternalTamper2EventCallback : RTC InternalTamper 2 Event callback.
- InternalTamper3EventCallback : RTC InternalTamper 3 Event callback.
- InternalTamper5EventCallback : RTC InternalTamper 5 Event callback.
- InternalTamper8EventCallback : RTC InternalTamper 8 Event callback. #if defined (__ARM_FEATURE_CMSE) && (__ARM_FEATURE_CMSE == 3U)
- AlarmAEventCallback_S : RTC Alarm A Event callback_S
- AlarmBEventCallback_S : RTC Alarm B Event callback_S.
- TimeStampEventCallback_S : RTC TimeStampEvent callback_S.
- WakeUpTimerEventCallback_S : RTC WakeUpTimerEvent callback_S.
- Tamper1EventCallback_S : RTC Tamper 1 Event callback_S.
- Tamper2EventCallback_S : RTC Tamper 2 Event callback_S.
- Tamper3EventCallback_S : RTC Tamper 3 Event callback_S.
- Tamper4EventCallback_S : RTC Tamper 4 Event callback_S.
- Tamper5EventCallback_S : RTC Tamper 5 Event callback_S.
- Tamper6EventCallback_S : RTC Tamper 6 Event callback_S.
- Tamper7EventCallback_S : RTC Tamper 7 Event callback_S.
- Tamper8EventCallback_S : RTC Tamper 8 Event callback_S.
- InternalTamper1EventCallback_S : RTC InternalTamper 1 Event callback_S.
- InternalTamper2EventCallback_S : RTC InternalTamper 2 Event callback_S.
- InternalTamper3EventCallback_S : RTC InternalTamper 3 Event callback_S.
- InternalTamper5EventCallback_S : RTC InternalTamper 5 Event callback_S.
- InternalTamper8EventCallback_S : RTC InternalTamper 8 Event callback_S. #endif
- MspInitCallback : RTC MspInit callback.
- MspDeInitCallback : RTC MspDeInit callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_RTC_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_RTC_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- AlarmAEventCallback : RTC Alarm A Event callback.
- AlarmBEventCallback : RTC Alarm B Event callback.
- TimeStampEventCallback : RTC TimeStamp Event callback.
- WakeUpTimerEventCallback : RTC WakeUpTimer Event callback.
- Tamper1EventCallback : RTC Tamper 1 Event callback.
- Tamper2EventCallback : RTC Tamper 2 Event callback.
- Tamper3EventCallback : RTC Tamper 3 Event callback.
- Tamper4EventCallback : RTC Tamper 4 Event callback.
- Tamper5EventCallback : RTC Tamper 5 Event callback.
- Tamper6EventCallback : RTC Tamper 6 Event callback.
- Tamper7EventCallback : RTC Tamper 7 Event callback.
- Tamper8EventCallback : RTC Tamper 8 Event callback.
- InternalTamper1EventCallback : RTC Internal Tamper 1 Event callback.

- InternalTamper2EventCallback : RTC Internal Tamper 2 Event callback.
- InternalTamper3EventCallback : RTC Internal Tamper 3 Event callback.
- InternalTamper4EventCallback : RTC Internal Tamper 4 Event callback.
- InternalTamper5EventCallback : RTC Internal Tamper 5 Event callback.
- InternalTamper6EventCallback : RTC Internal Tamper 6 Event callback.
- InternalTamper8EventCallback : RTC Internal Tamper 8 Event callback. #if defined (__ARM_FEATURE_CMSE) && (__ARM_FEATURE_CMSE == 3U)
- AlarmAEventCallback_S : RTC Alarm A Event callback secure.
- AlarmBEventCallback_S : RTC Alarm B Event callback secure.
- TimeStampEventCallback_S : RTC TimeStamp Event callback secure.
- WakeUpTimerEventCallback_S : RTC WakeUpTimer Event callback secure.
- Tamper1EventCallback_S : RTC Tamper 1 Event callback secure.
- Tamper2EventCallback_S : RTC Tamper 2 Event callback secure.
- Tamper3EventCallback_S : RTC Tamper 3 Event callback secure.
- Tamper4EventCallback_S : RTC Tamper 4 Event callback secure.
- Tamper5EventCallback_S : RTC Tamper 5 Event callback secure.
- Tamper6EventCallback_S : RTC Tamper 6 Event callback secure.
- Tamper7EventCallback_S : RTC Tamper 7 Event callback secure.
- Tamper8EventCallback_S : RTC Tamper 8 Event callback secure.
- InternalTamper1EventCallback_S : RTC Internal Tamper 1 Event callback secure.
- InternalTamper2EventCallback_S : RTC Internal Tamper 2 Event callback secure.
- InternalTamper3EventCallback_S : RTC Internal Tamper 3 Event callback secure.
- InternalTamper4EventCallback_S : RTC Internal Tamper 4 Event callback secure.
- InternalTamper5EventCallback_S : RTC Internal Tamper 5 Event callback secure.
- InternalTamper6EventCallback_S : RTC Internal Tamper 6 Event callback secure.
- InternalTamper8EventCallback_S : RTC Internal Tamper 8 Event callback secure. #endif
- MspInitCallback : RTC MspInit callback.
- MspDeInitCallback : RTC MspDeInit callback.

By default, after the @ref HAL_RTC_Init() and when the state is HAL_RTC_STATE_RESET, all callbacks are set to the corresponding weak functions : examples @ref AlarmAEventCallback(), @ref TimeStampEventCallback(). Exception done for MspInit and MspDeInit callbacks that are reset to the legacy weak function in the @ref HAL_RTC_Init()/@ref HAL_RTC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, @ref HAL_RTC_Init()/@ref HAL_RTC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_RTC_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_RTC_STATE_READY or HAL_RTC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_RTC_RegisterCallback() before calling @ref HAL_RTC_DeInit() or @ref HAL_RTC_Init() function.

When The compilation define USE_HAL_RTC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

48.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   – A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
   – When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.

2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- ***HAL_RTC_Init***
- ***HAL_RTC_DeInit***
- ***HAL_RTC_MspInit***
- ***HAL_RTC_MspDeInit***

48.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- ***HAL_RTC_SetTime***
- ***HAL_RTC_GetTime***
- ***HAL_RTC_SetDate***
- ***HAL_RTC_GetDate***

48.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- ***HAL_RTC_SetAlarm***
- ***HAL_RTC_SetAlarm_IT***
- ***HAL_RTC_DeactivateAlarm***
- ***HAL_RTC_GetAlarm***
- ***HAL_RTC_AlarmIRQHandler***
- ***HAL_RTC_AlarmAEventCallback***
- ***HAL_RTC_PollForAlarmAEvent***

48.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- ***HAL_RTC_WaitForSynchro***

48.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- ***HAL_RTC_GetState***

48.2.11 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This function does not reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name

void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_MspDeInit

Function name

void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_SetTime

Function name

HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Set RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_GetTime

Function name

HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name

HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Set RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_GetDate

Function name

HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Get RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to Date structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_SetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_SetAlarm_IT

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm with Interrupt.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

HAL_RTC_DeactivateAlarm

Function name

HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function description

Deactivate the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  - RTC_ALARM_A: AlarmA
  - RTC_ALARM_B: AlarmB

Return values

- **HAL:** status

HAL_RTC_GetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)

Function description

Get the RTC Alarm value and masks.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Date structure
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  - RTC_ALARM_A: AlarmA
  - RTC_ALARM_B: AlarmB
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name

void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Alarm interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle AlarmA Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTC_AlarmAEventCallback

Function name

void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm A callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_WaitForSynchro

Function name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name

HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)

Function description

Return the RTC handle state.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** state

RTC_EnterInitMode

Function name

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

Enter the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ExitInitMode

Function name

HAL_StatusTypeDef RTC_ExitInitMode (RTC_HandleTypeDef * hrtc)

Function description

Exit the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

RTC_ByteToBcd2

Function name

uint8_t RTC_ByteToBcd2 (uint8_t Value)

Function description

Convert a 2 digit decimal to BCD format.

Parameters

- **Value:** Byte to be converted

Return values

- **Converted:** byte

RTC_Bcd2ToByte

Function name

uint8_t RTC_Bcd2ToByte (uint8_t Value)

Function description

Convert from 2 digit BCD to Binary.

Parameters

- **Value:** BCD value to be converted

Return values

- **Converted:** word

48.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

48.3.1 RTC

RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC AlarmMask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:1] not used in Alarm comparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:2] not used in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:3] not used in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:4] not used in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:5] not used in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:6] not used in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:7] not used in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:8] not used in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:9] not used in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:10] not used in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:11] not used in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:12] not used in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14:13] not used in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDMASK_SS14

SS[14] not used in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDMASK_NONE

SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC Clear Flags Definitions

RTC_CLEAR_ITSF

Clear Internal Time-stamp flag

RTC_CLEAR_TSOVF

Clear Time-stamp overflow flag

RTC_CLEAR_TSF

Clear Time-stamp flag

RTC_CLEAR_WUTF

Clear Wakeup timer flag

RTC_CLEAR_ALRBF

Clear Alarm B flag

RTC_CLEAR_ALRAF

Clear Alarm A flag

RTC DayLightSaving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_ADD1H

Description:

- Add 1 hour (summer time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_SUB1H

Description:

- Subtract 1 hour (winter time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

__HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMA_DISABLE

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_ENABLE

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarms flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF
  - RTC_FLAG_ALRAWF
  - RTC_FLAG_ALRBWF

Return value:

- None

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarms pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_RISING_IT

Description:

- set rising edge interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_FALLING_IT

Description:

- set rising edge interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_CLEAR_IT

Description:

- clear interrupt on the RTC Alarm associated Exti line.

Return value:

- None

RTC Flags Definitions

RTC_FLAG_RECALPF

Recalibration pending Flag

RTC_FLAG_INITF

Initialization flag

RTC_FLAG_RSF

Registers synchronization flag

RTC_FLAG_INITS

Initialization status flag

RTC_FLAG_SHPF

Shift operation pending flag

RTC_FLAG_WUTWF

Wakeup timer write flag

RTC_FLAG_ALRBWF

Alarm B write flag

RTC_FLAG_ALRAWF

Alarm A write flag

RTC_FLAG_ITSF

Internal Time-stamp flag

RTC_FLAG_TSOVF

Time-stamp overflow flag

RTC_FLAG_TSF

Time-stamp flag

RTC_FLAG_WUTF

Wakeup timer flag

RTC_FLAG_ALRBF

Alarm B flag

RTC_FLAG_ALRAF

Alarm A flag

RTC Flag Mask (5bits) describe in RTC_Flags_Definitions

RTC_FLAG_MASK

RTC flags mask (5bits)

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

RTC_IT_TS

Enable Timestamp Interrupt

RTC_IT_WUT

Enable Wakeup timer Interrupt

RTC_IT_ALRA

Enable Alarm A Interrupt

RTC_IT_ALRB

Enable Alarm B Interrupt

RTC Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_PULLUP

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Pull-Up ALARM OUT

RTC_OUTPUT_PULLUP_NONE

RTC_OUTPUT_PULLUP_ON

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_PUSHPULL

RTC_OUTPUT_TYPE_OPENDRAIN

RTC StoreOperation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

49 HAL RTC Extension Driver

49.1 RTCEx Firmware driver registers structures

49.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32g4xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger***
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the TAMP Filter Tamper. This parameter can be a value of ***RTCEx_Tamper_Filter***
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of ***RTCEx_Tamper_Sampling_Frequencies***
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration . This parameter can be a value of ***RTCEx_Tamper_Pin_Precharge_Duration***
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp . This parameter can be a value of ***RTCEx_Tamper_Pull_UP***
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***

49.1.2 RTC_InternalTamperTypeDef

RTC_InternalTamperTypeDef is defined in the stm32g4xx_hal_rtc_ex.h

Data Fields

- ***uint32_t IntTamper***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_InternalTamperTypeDef::IntTamper***
  Specifies the Internal Tamper Pin. This parameter can be a value of ***RTCEx_Internal_Tamper_Pins***

- ***uint32_t RTC_InternalTamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***

49.2 RTCEx Firmware driver API description

The following section lists the various functions of the RTCEx library.

49.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal, use the HAL_RTC_Init() function.
- RTC_CALIB: this output is 512Hz signal or 1Hz. To enable the RTC_CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- Two pins can be used as RTC_ALARM or RTC_CALIB (PC13, PB2) managed on the RTC_OR register.
- When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured in output alternate function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEx_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC_TAMPCR register.
- If you do not intend to have tamper using RTC clock, you can bypass its initialization by setting ClockEnable init field to RTC_CLOCK_DISABLE.

- Enable Internal tamper using HAL_RTCEx_SetInternalTamper. IT mode can be chosen using setting Interrupt field.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEx_BKUPRead() function.

49.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- ***HAL_RTCEx_SetTimeStamp***
- ***HAL_RTCEx_SetTimeStamp_IT***
- ***HAL_RTCEx_DeactivateTimeStamp***
- ***HAL_RTCEx_SetInternalTimeStamp***
- ***HAL_RTCEx_DeactivateInternalTimeStamp***
- ***HAL_RTCEx_GetTimeStamp***
- ***HAL_RTCEx_TimeStampEventCallback***
- ***HAL_RTCEx_TimeStampIRQHandler***
- ***HAL_RTCEx_PollForTimeStampEvent***

49.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- ***HAL_RTCEx_SetWakeUpTimer***
- ***HAL_RTCEx_SetWakeUpTimer_IT***
- ***HAL_RTCEx_DeactivateWakeUpTimer***
- ***HAL_RTCEx_GetWakeUpTimer***
- ***HAL_RTCEx_WakeUpTimerIRQHandler***
- ***HAL_RTCEx_WakeUpTimerEventCallback***
- ***HAL_RTCEx_PollForWakeUpTimerEvent***

49.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- ***HAL_RTCEx_SetSmoothCalib***
- ***HAL_RTCEx_SetSynchroShift***
- ***HAL_RTCEx_SetCalibrationOutPut***
- ***HAL_RTCEx_DeactivateCalibrationOutPut***

- *HAL_RTCEx_SetRefClock*
- *HAL_RTCEx_DeactivateRefClock*
- *HAL_RTCEx_EnableBypassShadow*
- *HAL_RTCEx_DisableBypassShadow*

49.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- *HAL_RTCEx_AlarmBEventCallback*
- *HAL_RTCEx_PollForAlarmBEvent*

49.2.6 Tamper functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that ine you can select to output tamper event on RTC pin.
- Enable the Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP, timestamp using the HAL_RTCEx_SetTamper() function. You can configure Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the TAMP_TAMPCR register.
- Enable Internal Tamper and configure it with interrupt, timestamp using the HAL_RTCEx_SetInternalTamper() function.

This section contains the following APIs:

- *HAL_RTCEx_SetTamper*
- *HAL_RTCEx_SetTamper_IT*
- *HAL_RTCEx_DeactivateTamper*
- *HAL_RTCEx_PollForTamperEvent*
- *HAL_RTCEx_SetInternalTamper*
- *HAL_RTCEx_SetInternalTamper_IT*
- *HAL_RTCEx_DeactivateInternalTamper*
- *HAL_RTCEx_PollForInternalTamperEvent*
- *HAL_RTCEx_TamperIRQHandler*
- *HAL_RTCEx_Tamper1EventCallback*
- *HAL_RTCEx_Tamper2EventCallback*
- *HAL_RTCEx_Tamper3EventCallback*
- *HAL_RTCEx_InternalTamper3EventCallback*
- *HAL_RTCEx_InternalTamper4EventCallback*
- *HAL_RTCEx_InternalTamper5EventCallback*
- *HAL_RTCEx_InternalTamper6EventCallback*

49.2.7 Extended RTC Backup register functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that ine you can select to output tamper event on RTC pin.

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register

This section contains the following APIs:

- *HAL_RTCEx_BKUPWrite*
- *HAL_RTCEx_BKUPRead*

49.2.8 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp with Interrupt.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_DeactivateTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Set Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This API must be called before enabling the internal TimeStamp feature.

HAL_RTCEx_DeactivateInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function description

Get the RTC TimeStamp value.

Parameters

- **hrtc:** RTC handle
- **sTimeStamp:** Pointer to Time structure
- **sTimeStampDate:** Pointer to Date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTCEx_TimeStampIRQHandler

Function name

void HAL_RTCEx_TimeStampIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle TimeStamp interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForTimeStampEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle TimeStamp polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_TimeStampEventCallback

Function name

void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)

Function description

TimeStamp callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer with interrupt.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_DeactivateWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Deactivate wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Get wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Wake Up Timer interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name

void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Wake Up Timer callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Wake Up Timer Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetSmoothCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)

Function description

Set the Smooth calibration parameters.

Parameters

- **hrtc:** RTC handle
- **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be can be one of the following values :
  - RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.
  - RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.
  - RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.
- **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:
  - RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.
  - RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.
- **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetSynchroShift

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)

Function description

Configure the Synchronization Shift Control Settings.

Parameters

- **hrtc:** RTC handle
- **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values:
  - RTC_SHIFTADD1S_SET: Add one second to the clock calendar.
  - RTC_SHIFTADD1S_RESET: No effect.
- **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.

Return values

- **HAL:** status

Notes

- When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function description

Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle
- **CalibOutput:** Select the Calibration output Selection . This parameter can be one of the following values:
  - RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.
  - RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)

Function description

Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)

Function description

Enable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

Disable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Enable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Disable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_AlarmBEventCallback

Function name

void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm B callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForAlarmBEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Alarm B Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function description

Set Tamper.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function description

Set Tamper in IT mode.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)

Function description

Deactivate Tamper.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be a combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2

Return values

- **HAL:** status

HAL_RTCEx_PollForTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamperEvent (RTC_HandleTypeDef * hrtc, uint32_t Tamper, uint32_t Timeout)

Function description

Tamper event polling.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be a combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper (RTC_HandleTypeDef * hrtc, RTC_InternalTamperTypeDef * sIntTamper)

Function description

Set Internal Tamper in interrupt mode.

Parameters

- **hrtc:** RTC handle
- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper_IT (RTC_HandleTypeDef * hrtc, RTC_InternalTamperTypeDef * sIntTamper)

Function description

Set Internal Tamper.

Parameters

- **hrtc:** RTC handle
- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTamper (RTC_HandleTypeDef * hrtc, uint32_t IntTamper)

Function description

Deactivate Internal Tamper.

Parameters

- **hrtc:** RTC handle
- **IntTamper:** Selected internal tamper event. This parameter can be any combination of existing internal tampers.

Return values

- **HAL:** status

HAL_RTCEx_PollForInternalTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForInternalTamperEvent (RTC_HandleTypeDef * hrtc, uint32_t IntTamper, uint32_t Timeout)

Function description

Internal Tamper event polling.

Parameters

- **hrtc:** RTC handle
- **IntTamper:** selected tamper. This parameter can be any combination of existing internal tampers.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_TamperIRQHandler

Function name

void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Tamper interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper1EventCallback

Function name

void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper2EventCallback

Function name

void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper3EventCallback

Function name

void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper3EventCallback

Function name

void HAL_RTCEx_InternalTamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper4EventCallback

Function name

void HAL_RTCEx_InternalTamper4EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 4 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper5EventCallback

Function name

void HAL_RTCEx_InternalTamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 5 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper6EventCallback

Function name

void HAL_RTCEx_InternalTamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 6 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPWrite

Function name

void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)

Function description

Write a data in a specified TAMP Backup data register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.
- **Data:** Data to be written in the specified TAMP Backup data register.

Return values

- **None:**

HAL_RTCEx_BKUPRead

Function name

uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)

Function description

Reads data from the specified TAMP Backup data Register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to RTC_BACKUP_NB to specify the register.

Return values

- **Read:** value

49.3 RTCEx Firmware driver defines

The following section lists the various define and macros of the module.

49.3.1 RTCEx

RTCEx

RTCEx Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTCEx Backup Registers Definition

RTC_BKP_NUMBER

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTC Calibration

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:

- Enable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:

- Disable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operations flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- None

RTCEx Calib Output selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Exported Macros

__HAL_RTC_CLEAR_FLAG

Description:

- Clear the specified RTC pending flag.

Parameters:

- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_CLEAR_ITSF Clear Internal Time-stamp flag
  - RTC_CLEAR_TSOVF Clear Time-stamp overflow flag
  - RTC_CLEAR_TSF Clear Time-stamp flag
  - RTC_CLEAR_WUTF Clear Wakeup timer flag
  - RTC_CLEAR_ALRBF Clear Alarm B flag
  - RTC_CLEAR_ALRAF Clear Alarm A flag

Return value:

- None

__HAL_RTC_GET_FLAG

Description:

- Check whether the specified RTC flag is set or not.

Parameters:

- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_FLAG_RECALPF Recalibration pending Flag
  - RTC_FLAG_INITF Initialization flag
  - RTC_FLAG_RSF Registers synchronization flag
  - RTC_FLAG_INITS Initialization status flag
  - RTC_FLAG_SHPF Shift operation pending flag
  - RTC_FLAG_WUTWF Wakeup timer write flag
  - RTC_FLAG_ALRBWF Alarm B write flag
  - RTC_FLAG_ALRAWF Alarm A write flag
  - RTC_FLAG_ITSF Internal Time-stamp flag
  - RTC_FLAG_TSOVF Time-stamp overflow flag
  - RTC_FLAG_TSF Time-stamp flag
  - RTC_FLAG_WUTF Wakeup timer flag
  - RTC_FLAG_ALRBF Alarm B flag
  - RTC_FLAG_ALRAF Alarm A flag

Return value:

- None

RTCEx Flags

RTC_FLAG_TAMP_1

RTC_FLAG_TAMP_2

RTC_FLAG_TAMP_3

RTC_FLAG_TAMP_ALL

RTC_FLAG_INT_TAMP_1

RTC_FLAG_INT_TAMP_2

RTC_FLAG_INT_TAMP_3

Tamper 3 Interrupt flag

RTC_FLAG_INT_TAMP_4

Tamper 4 Interrupt flag

RTC_FLAG_INT_TAMP_5

Tamper 5 Interrupt flag

RTC_FLAG_INT_TAMP_6

Tamper 6 Interrupt flag

RTC_FLAG_INT_TAMP_7

RTC_FLAG_INT_TAMP_8

RTC_FLAG_INT_TAMP_ALL

RTCEx Internal Tamper Interrupt

RTC_IT_TAMP_1

Tamper 1 Interrupt

RTC_IT_TAMP_2

Tamper 2 Interrupt

RTC_IT_TAMP_3

Tamper 3 Interrupt

RTC_IT_TAMP_ALL

RTC_IT_INT_TAMP_1

RTC_IT_INT_TAMP_2

RTC_IT_INT_TAMP_3

Tamper 3 internal Interrupt

RTC_IT_INT_TAMP_4

Tamper 4 internal Interrupt

RTC_IT_INT_TAMP_5

Tamper 5 internal Interrupt

RTC_IT_INT_TAMP_6

Tamper 6 internal Interrupt

RTC_IT_INT_TAMP_7

RTC_IT_INT_TAMP_8

RTC_IT_INT_TAMP_ALL

RTCEx Internal Tamper Pins Definition

RTC_INT_TAMPER_1

RTC_INT_TAMPER_2

RTC_INT_TAMPER_3

RTC_INT_TAMPER_4

RTC_INT_TAMPER_5

RTC_INT_TAMPER_6

RTC_INT_TAMPER_7

RTC_INT_TAMPER_8

RTC_INT_TAMPER_ALL

Private macros to check input parameters

IS_TIMESTAMP_EDGE

IS_RTC_TIMESTAMP_PIN

IS_RTC_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_TAMPERDETECTIONOUTPUT

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_LOW_POWER_CALIB

IS_RTC_TAMPER

IS_RTC_INTERNAL_TAMPER

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_BKP

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTC_OUTPUT_TAMPER

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

RTCEx Smooth calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

RTCEx tamper

__HAL_RTC_TAMPER_ENABLE

Description:

- Enable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper source to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2

Return value:

- None

__HAL_RTC_TAMPER_DISABLE

Description:

- Disable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper sources to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- Disable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

- Check whether the specified TAMP Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified TAMP Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected TAMP Tampers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag

Return value:

- None

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the TAMP Tampers pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag

Return value:

- None

__HAL_RTC_TAMPER_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Tamper associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Tamper associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_EXTI_RISING_IT

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_EXTI_FALLING_IT

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_EXTI_CLEAR_IT

Description:

- Clear the interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Tamper associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Tamper associated Exti line.

Return value:

- None

RTCEx Tamper EraseBackUp

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTCEx Tamper Filter

RTC_TAMPERFILTER_DISABLE

Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level

RTCEx Tamper MaskFlag

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTCEx Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTC_TAMPER_ALL

RTCEx Tamper Pin Precharge Duration

RTC_TAMPERPRECHARGEDURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx Tamper Pull UP

RTC_TAMPER_PULLUP_ENABLE

Tamper pins are pre-charged before sampling

RTC_TAMPER_PULLUP_DISABLE

Tamper pins pre-charge is disabled

RTCEx Tamper Sampling Frequencies

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTCEx Tamper TimeStamp On Tamper Detection

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

TimeStamp on Tamper Detection event is not saved

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE

TimeStamp on Tamper Detection event saved

RTCEx Tamper Trigger

RTC_TAMPERTRIGGER_RISINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_FALLINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_LOWLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_HIGHLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC Timestamp

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamps flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamps pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag to clear. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_RISING_IT

Description:

- set the rising edge for interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_FALLING_IT

Description:

- set the falling edge for interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_CLEAR_IT

Description:

- Clear the interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_CLEAR_FLAG

Description:

- Clear the interrupt on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE

Description:

- Enable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE

Description:

- Disable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC Internal Time Stamps flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Internal Time Stamps pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_TAMPTS_ENABLE

Description:

- Enable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPTS_DISABLE

Description:

- Disable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPOE_ENABLE

Description:

- Enable the RTC Tamper detection output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPOE_DISABLE

Description:

- Disable the RTC Tamper detection output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC WakeUp Timer

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
  - RTC_FLAG_WUTF
  - RTC_FLAG_WUTWF

Return value:

- None

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timers pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - RTC_FLAG_WUTF

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_RISING_IT

Description:

- set the rising edge for interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_FALLING_IT

Description:

- set the falling edge for interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_IT

Description:

- Clear the interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG

Description:

- Clear the interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

RTCEx Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

50 HAL SAI Generic Driver

50.1 SAI Firmware driver registers structures

50.1.1 SAI_PdmInitTypeDef

SAI_PdmInitTypeDef is defined in the stm32g4xx_hal_sai.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t MicPairsNbr***
- ***uint32_t ClockEnable***

Field Documentation

- ***FunctionalState SAI_PdmInitTypeDef::Activation***
  Enable/disable PDM interface
- ***uint32_t SAI_PdmInitTypeDef::MicPairsNbr***
  Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAI_PdmInitTypeDef::ClockEnable***
  Specifies which clock must be enabled. This parameter can be a values combination of ***SAI_PDM_ClockEnable***

50.1.2 SAI_InitTypeDef

SAI_InitTypeDef is defined in the stm32g4xx_hal_sai.h

Data Fields

- ***uint32_t AudioMode***
- ***uint32_t Synchro***
- ***uint32_t SynchroExt***
- ***uint32_t MckOutput***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t AudioFrequency***
- ***uint32_t Mckdiv***
- ***uint32_t MckOverSampling***
- ***uint32_t MonoStereoMode***
- ***uint32_t CompandingMode***
- ***uint32_t TriState***
- ***SAI_PdmInitTypeDef PdmInit***
- ***uint32_t Protocol***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***

Field Documentation

- ***uint32_t SAI_InitTypeDef::AudioMode***
  Specifies the SAI Block audio Mode. This parameter can be a value of ***SAI_Block_Mode***
- ***uint32_t SAI_InitTypeDef::Synchro***
  Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***

- ***uint32_t SAI_InitTypeDef::SynchroExt***

  Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of ***SAI_Block_SyncExt***

  Note:

  – If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block
- ***uint32_t SAI_InitTypeDef::MckOutput***

  Specifies whether master clock output will be generated or not. This parameter can be a value of ***SAI_Block_MckOutput***
- ***uint32_t SAI_InitTypeDef::OutputDrive***

  Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***

  Note:

  – This value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32_t SAI_InitTypeDef::NoDivider***

  Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***

  Note:

  – If bit NODIV in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI_xCR1 register is set, the frame length can take any of the values from 8 to 256.
- ***uint32_t SAI_InitTypeDef::FIFOThreshold***

  Specifies SAI Block FIFO threshold. This parameter can be a value of ***SAI_Block_Fifo_Threshold***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***

  Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***
- ***uint32_t SAI_InitTypeDef::Mckdiv***

  Specifies the master clock divider. This parameter must be a number between Min_Data = 0 and Max_Data = 63.

  Note:

  – This parameter is used only if AudioFrequency is set to SAI_AUDIO_FREQUENCY_MCKDIV otherwise it is internally computed.
- ***uint32_t SAI_InitTypeDef::MckOverSampling***

  Specifies the master clock oversampling. This parameter can be a value of ***SAI_Block_Mck_OverSampling***
- ***uint32_t SAI_InitTypeDef::MonoStereoMode***

  Specifies if the mono or stereo mode is selected. This parameter can be a value of ***SAI_Mono_Stereo_Mode***
- ***uint32_t SAI_InitTypeDef::CompandingMode***

  Specifies the companding mode type. This parameter can be a value of ***SAI_Block_Companding_Mode***
- ***uint32_t SAI_InitTypeDef::TriState***

  Specifies the companding mode type. This parameter can be a value of ***SAI_TRIState_Management***
- ***SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit***

  Specifies the PDM configuration.
- ***uint32_t SAI_InitTypeDef::Protocol***

  Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***uint32_t SAI_InitTypeDef::DataSize***

  Specifies the SAI Block data size. This parameter can be a value of ***SAI_Block_Data_Size***
- ***uint32_t SAI_InitTypeDef::FirstBit***

  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SAI_Block_MSB_LSB_transmission***

- ***uint32_t SAI_InitTypeDef::ClockStrobing***

  Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of ***SAI_Block_Clock_Strobing***

50.1.3 SAI_FrameInitTypeDef

SAI_FrameInitTypeDef is defined in the stm32g4xx_hal_sai.h

Data Fields

- ***uint32_t FrameLength***
- ***uint32_t ActiveFrameLength***
- ***uint32_t FSDefinition***
- ***uint32_t FSPolarity***
- ***uint32_t FSOffset***

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***

  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.

  Note:

  – If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.

- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***

  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***

  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***

- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***

  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***

- ***uint32_t SAI_FrameInitTypeDef::FSOffset***

  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

50.1.4 SAI_SlotInitTypeDef

SAI_SlotInitTypeDef is defined in the stm32g4xx_hal_sai.h

Data Fields

- ***uint32_t FirstBitOffset***
- ***uint32_t SlotSize***
- ***uint32_t SlotNumber***
- ***uint32_t SlotActive***

Field Documentation

- ***uint32_t SAI_SlotInitTypeDef::FirstBitOffset***

  Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24

- ***uint32_t SAI_SlotInitTypeDef::SlotSize***

  Specifies the Slot Size. This parameter can be a value of ***SAI_Block_Slot_Size***

- ***uint32_t SAI_SlotInitTypeDef::SlotNumber***

  Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16

- ***uint32_t SAI_SlotInitTypeDef::SlotActive***

  Specifies the slots in audio frame that will be activated. This parameter can be a value of ***SAI_Block_Slot_Active***

50.1.5 __SAI_HandleTypeDef

__SAI_HandleTypeDef is defined in the stm32g4xx_hal_sai.h

Data Fields

- ***SAI_Block_TypeDef * Instance***
- ***SAI_InitTypeDef Init***
- ***SAI_FrameInitTypeDef FrameInit***
- ***SAI_SlotInitTypeDef SlotInit***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***SAIcallback mutecallback***
- ***void(* InterruptServiceRoutine***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SAI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size
- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object
- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code

50.2 SAI Firmware driver API description

The following section lists the various functions of the SAI library.

50.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
   a. Enable the SAI interface clock.
   b. SAI pins configuration:
      - Enable the clock for the SAI GPIOs.
      - Configure these SAI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
      - Configure the SAI interrupt priority.
      - Enable the NVIC SAI IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA() and HAL_SAI_Receive_DMA() APIs):
      - Declare a DMA handle structure for the Tx/Rx stream.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream.
      - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
   a. Expert mode : Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that SAI clock source is configured:

- *SYSCLK or*
- *PLLQ output or*
- *HSI or*
- *External clock source which is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32g4xx_hal_conf.h file.*

Note: In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.

Note: In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

Note: It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- *First bit Offset <= (SLOT size - Data size)*
- *Data size <= SLOT size*
- *Number of SLOT x SLOT size = Frame length*
- *The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.*

Note: PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAI1 sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()

- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver :

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver :

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts
- __HAL_SAI_DISABLE_IT(): Disable the specified SAI interrupts
- __HAL_SAI_GET_IT_SOURCE(): Check if the specified SAI interrupt source is enabled or disabled
- __HAL_SAI_GET_FLAG(): Check whether the specified SAI flag is set or not

Callback registration

The compilation define USE_HAL_SAI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_SAI_RegisterCallback() to register a user callback.

Function HAL_SAI_RegisterCallback() allows to register following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_SAI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_SAI_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

By default, after the HAL_SAI_Init and if the state is HAL_SAI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_SAI_RxCpltCallback(), HAL_SAI_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_SAI_Init and HAL_SAI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SAI_Init and HAL_SAI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SAI_RegisterCallback before calling HAL_SAI_DeInit or HAL_SAI_Init function.

When the compilation define USE_HAL_SAI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

50.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL_SAI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SAI_Init() to configure the selected device with the selected configuration:
  - Mode (Master/slave TX/RX)
  - Protocol
  - Data Size
  - MCLK Output
  - Audio frequency
  - FIFO Threshold
  - Frame Config
  - Slot Config
  - PDM Config
- Call the function HAL_SAI_DeInit() to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

- ***HAL_SAI_InitProtocol***
- ***HAL_SAI_Init***
- ***HAL_SAI_DeInit***
- ***HAL_SAI_MspInit***
- ***HAL_SAI_MspDeInit***

50.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SAI data transfers.

- There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
  - HAL_SAI_Transmit()
  - HAL_SAI_Receive()
- Non Blocking mode functions with Interrupt are :
  - HAL_SAI_Transmit_IT()
  - HAL_SAI_Receive_IT()
- Non Blocking mode functions with DMA are :
  - HAL_SAI_Transmit_DMA()
  - HAL_SAI_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SAI_TxCpltCallback()
  - HAL_SAI_RxCpltCallback()
  - HAL_SAI_ErrorCallback()

This section contains the following APIs:

- ***HAL_SAI_Transmit***
- ***HAL_SAI_Receive***
- ***HAL_SAI_Transmit_IT***
- ***HAL_SAI_Receive_IT***
- ***HAL_SAI_DMAPause***
- ***HAL_SAI_DMAResume***
- ***HAL_SAI_DMAStop***
- ***HAL_SAI_Abort***
- ***HAL_SAI_Transmit_DMA***
- ***HAL_SAI_Receive_DMA***
- ***HAL_SAI_EnableTxMuteMode***
- ***HAL_SAI_DisableTxMuteMode***
- ***HAL_SAI_EnableRxMuteMode***
- ***HAL_SAI_DisableRxMuteMode***
- ***HAL_SAI_IRQHandler***
- ***HAL_SAI_TxCpltCallback***
- ***HAL_SAI_TxHalfCpltCallback***
- ***HAL_SAI_RxCpltCallback***
- ***HAL_SAI_RxHalfCpltCallback***
- ***HAL_SAI_ErrorCallback***

50.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_SAI_GetState***
- ***HAL_SAI_GetError***

50.2.5 Detailed description of functions

HAL_SAI_InitProtocol

Function name

HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)

Function description

Initialize the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **protocol:** one of the supported protocol SAI Supported protocol
- **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module.
- **nbslot:** Number of slot.

Return values

- **HAL:** status

HAL_SAI_Init

Function name

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI according to the specified parameters.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DeInit

Function name

HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI peripheral.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_MspInit

Function name

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_MspDeInit

Function name

void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_Transmit

Function name

HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Receive

Function name

HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_DMAPause

Function name

HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)

Function description

Pause the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAResume

Function name

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

Resume the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAStop

Function name

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

Stop the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_Abort

Function name

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

Abort the current transfer and disable the SAI.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

Enable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **val:** value sent during the mute SAI Block Mute Value

Return values

- **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)

Function description

Enable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback:** function called when the mute is detected.
- **counter:** number a data before mute detection max 63.

Return values

- **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_IRQHandler

Function name

void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)

Function description

Handle SAI interrupt request.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer Half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_ErrorCallback

Function name

void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)

Function description

SAI error callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_GetState

Function name

HAL_SAI_StateTypeDef HAL_SAI_GetState (SAI_HandleTypeDef * hsai)

Function description

Return the SAI handle state.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** state

HAL_SAI_GetError

Function name

uint32_t HAL_SAI_GetError (SAI_HandleTypeDef * hsai)

Function description

Return the SAI error code.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- **SAI:** Error Code

50.3 SAI Firmware driver defines

The following section lists the various define and macros of the module.

50.3.1 SAI

SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

SAI_FIFOSTATUS_LESS1QUARTERFULL

SAI_FIFOSTATUS_1QUARTERFULL

SAI_FIFOSTATUS_HALFFULL

SAI_FIFOSTATUS_3QUARTERFULL

SAI_FIFOSTATUS_FULL

SAI Block Fifo Threshold

SAI_FIFOTHRESHOLD_EMPTY

SAI_FIFOTHRESHOLD_1QF

SAI_FIFOTHRESHOLD_HF

SAI_FIFOTHRESHOLD_3QF

SAI_FIFOTHRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR

SAI_FLAG_MUTEDET

SAI_FLAG_WCKCFG

SAI_FLAG_FREQ

SAI_FLAG_CNRDY

SAI_FLAG_AFSDET

SAI_FLAG_LFSDET

SAI Block FS Definition

SAI_FS_STARTFRAME

SAI_FS_CHANNEL_IDENTIFICATION

SAI Block FS Offset

SAI_FS_FIRSTBIT

SAI_FS_BEFOREFIRSTBIT

SAI Block FS Polarity

SAI_FS_ACTIVE_LOW

SAI_FS_ACTIVE_HIGH

SAI Block Interrupts Definition

SAI_IT_OVRUDR

SAI_IT_MUTEDET

SAI_IT_WCKCFG

SAI_IT_FREQ

SAI_IT_CNRDY

SAI_IT_AFSDET

SAI_IT_LFSDET

SAI Block Master Clock Output

SAI_MCK_OUTPUT_DISABLE

SAI_MCK_OUTPUT_ENABLE

SAI Block Master Clock OverSampling

SAI_MCK_OVERSAMPLING_DISABLE

SAI_MCK_OVERSAMPLING_ENABLE

SAI Block Mode

SAI_MODEMASTER_TX

SAI_MODEMASTER_RX

SAI_MODESLAVE_TX

SAI_MODESLAVE_RX

SAI Block MSB LSB transmission

SAI_FIRSTBIT_MSB

SAI_FIRSTBIT_LSB

SAI Block Mute Value

SAI_ZERO_VALUE

SAI_LAST_SENT_VALUE

SAI Block NoDivider

SAI_MASTERDIVIDER_ENABLE

SAI_MASTERDIVIDER_DISABLE

SAI Block Output Drive

SAI_OUTPUTDRIVE_DISABLE

SAI_OUTPUTDRIVE_ENABLE

SAI Block Protocol

SAI_FREE_PROTOCOL

SAI_SPDIF_PROTOCOL

SAI_AC97_PROTOCOL

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCHRONOUS

Asynchronous

SAI_SYNCHRONOUS

Synchronous with other block of same SAI

SAI_SYNCHRONOUS_EXT_SAI1

Synchronous with other SAI, SAI1

SAI_SYNCHRONOUS_EXT_SAI2

Synchronous with other SAI, SAI2

SAI Error Code

HAL_SAI_ERROR_NONE

No error

HAL_SAI_ERROR_OVR

Overrun Error

HAL_SAI_ERROR_UDR

Underrun error

HAL_SAI_ERROR_AFSDET

Anticipated Frame synchronisation detection

HAL_SAI_ERROR_LFSDET

Late Frame synchronisation detection

HAL_SAI_ERROR_CNREADY

codec not ready

HAL_SAI_ERROR_WCKCFG

Wrong clock configuration

HAL_SAI_ERROR_TIMEOUT

Timeout error

HAL_SAI_ERROR_DMA

DMA error

SAI Exported Macros

__HAL_SAI_RESET_HANDLE_STATE

Description:

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT

Description:

- Enable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

Description:

- Disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_GET_IT_SOURCE

Description:

- Check whether the specified SAI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_SAI_GET_FLAG

Description:

- Check whether the specified SAI flag is set or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SAI_FLAG_OVRUDR: Overrun underrun flag.
  - SAI_FLAG_MUTEDET: Mute detection flag.
  - SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
  - SAI_FLAG_FREQ: FIFO request flag.
  - SAI_FLAG_CNRDY: Codec not ready flag.
  - SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
  - SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI_FLAG_OVRUDR: Clear Overrun underrun
  - SAI_FLAG_MUTEDET: Clear Mute detection
  - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
  - SAI_FLAG_FREQ: Clear FIFO request
  - SAI_FLAG_CNRDY: Clear Codec not ready
  - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
  - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

Description:

- Enable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_DISABLE

Description:

- Disable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

SAI Mono Stereo Mode

SAI_STEREOMODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIState Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

51 HAL SAI Extension Driver

51.1 SAIEx Firmware driver registers structures

51.1.1 SAIEx_PdmMicDelayParamTypeDef

SAIEx_PdmMicDelayParamTypeDef is defined in the stm32g4xx_hal_sai_ex.h

Data Fields

- ***uint32_t MicPair***
- ***uint32_t LeftDelay***
- ***uint32_t RightDelay***

Field Documentation

- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair***

  Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay***

  Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay***

  Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

51.2 SAIEx Firmware driver API description

The following section lists the various functions of the SAIEx library.

51.2.1 Extended features functions

This section provides functions allowing to:

- Modify PDM microphone delays

This section contains the following APIs:

- ***HAL_SAIEx_ConfigPdmMicDelay***

51.2.2 Detailed description of functions

HAL_SAIEx_ConfigPdmMicDelay

Function name

HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (SAI_HandleTypeDef * hsai, SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)

Function description

Configure PDM microphone delays.

Parameters

- **hsai:** SAI handle.
- **pdmMicDelay:** Microphone delays configuration.

Return values

- **HAL:** status

52 HAL SMARTCARD Generic Driver

52.1 SMARTCARD Firmware driver registers structures

52.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32g4xx_hal_smartcard.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***

  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hsmartcard->Init.BaudRate))) where usart_ker_ckpres is the USART input clock divided by a prescaler

- ***uint32_t SMARTCARD_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).

- ***uint32_t SMARTCARD_InitTypeDef::StopBits***

  Specifies the number of stop bits. This parameter can be a value of ***SMARTCARD_Stop_Bits***.

- ***uint16_t SMARTCARD_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***

  Note:

  – The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.

- ***uint16_t SMARTCARD_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***

- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***

  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***

- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***

  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***

- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***

  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***
- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***

  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***

  Specifies the SmartCard Prescaler. This parameter can be any value from 0x01 to 0x1F. Prescaler value is multiplied by 2 to give the division factor of the source clock frequency
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***

  Specifies the SmartCard Guard Time applied after stop bits.
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***

  Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***

  Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***

  Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***

  Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***

  Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)
- ***uint32_t SMARTCARD_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***SMARTCARD_ClockPrescaler***.

52.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32g4xx_hal_smartcard.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***
- ***uint16_t TxCompletionIndication***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***

  Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of ***SMARTCARDEx_Advanced_Features_Initialization_Type***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***

  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***SMARTCARD_Rx_Tx_Swap***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***SMARTCARD_Overrun_Disable***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***SMARTCARD_DMA_Disable_on_Rx_Error***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***SMARTCARD_MSB_First***
- ***uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication***
  Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of ***SMARTCARDEx_Transmission_Completion_Indication***.

52.1.3 __SMARTCARD_HandleTypeDef

__SMARTCARD_HandleTypeDef is defined in the stm32g4xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef __SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters

- ***SMARTCARD_AdvFeatureInitTypeDef __SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***uint8_t* __SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* __SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***uint16_t __SMARTCARD_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __SMARTCARD_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __SMARTCARD_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***SMARTCARDEx_FIFO_mode***.
- ***void(* __SMARTCARD_HandleTypeDef::RxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __SMARTCARD_HandleTypeDef::TxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef __SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO uint32_t __SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code

52.2 SMARTCARD Firmware driver API description

The following section lists the various functions of the SMARTCARD library.

52.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.

3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note: The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()

- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note: You can refer to the SMARTCARD HAL driver header file for more useful macros

52.2.2 Callback registration

The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_SMARTCARD_RegisterCallback() to register a user callback. Function @ref HAL_SMARTCARD_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_SMARTCARD_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_SMARTCARD_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit.

By default, after the @ref HAL_SMARTCARD_Init() and when the state is HAL_SMARTCARD_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples @ref HAL_SMARTCARD_TxCpltCallback(), @ref HAL_SMARTCARD_RxCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_SMARTCARD_Init() and @ref HAL_SMARTCARD_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_SMARTCARD_Init() and @ref HAL_SMARTCARD_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_SMARTCARD_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_SMARTCARD_STATE_READY or HAL_SMARTCARD_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_SMARTCARD_RegisterCallback() before calling @ref HAL_SMARTCARD_DeInit() or @ref HAL_SMARTCARD_Init() function.

When The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

52.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_SMARTCARD_Init***
- ***HAL_SMARTCARD_DeInit***
- ***HAL_SMARTCARD_MspInit***
- ***HAL_SMARTCARD_MspDeInit***

52.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.

- There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
  - The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
- Blocking mode APIs are :
  - HAL_SMARTCARD_Transmit()
  - HAL_SMARTCARD_Receive()
- Non Blocking mode APIs with Interrupt are :
  - HAL_SMARTCARD_Transmit_IT()
  - HAL_SMARTCARD_Receive_IT()
  - HAL_SMARTCARD_IRQHandler()
- Non Blocking mode functions with DMA are :
  - HAL_SMARTCARD_Transmit_DMA()
  - HAL_SMARTCARD_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SMARTCARD_TxCpltCallback()
  - HAL_SMARTCARD_RxCpltCallback()
  - HAL_SMARTCARD_ErrorCallback()

1. Non-Blocking mode transfers could be aborted using Abort API's :
   - HAL_SMARTCARD_Abort()
   - HAL_SMARTCARD_AbortTransmit()
   - HAL_SMARTCARD_AbortReceive()
   - HAL_SMARTCARD_Abort_IT()
   - HAL_SMARTCARD_AbortTransmit_IT()
   - HAL_SMARTCARD_AbortReceive_IT()
2. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   - HAL_SMARTCARD_AbortCpltCallback()
   - HAL_SMARTCARD_AbortTransmitCpltCallback()
   - HAL_SMARTCARD_AbortReceiveCpltCallback()
3. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   - Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode tranmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_SMARTCARD_Transmit***
- ***HAL_SMARTCARD_Receive***
- ***HAL_SMARTCARD_Transmit_IT***
- ***HAL_SMARTCARD_Receive_IT***
- ***HAL_SMARTCARD_Transmit_DMA***

- *HAL_SMARTCARD_Receive_DMA*
- *HAL_SMARTCARD_Abort*
- *HAL_SMARTCARD_AbortTransmit*
- *HAL_SMARTCARD_AbortReceive*
- *HAL_SMARTCARD_Abort_IT*
- *HAL_SMARTCARD_AbortTransmit_IT*
- *HAL_SMARTCARD_AbortReceive_IT*
- *HAL_SMARTCARD_IRQHandler*
- *HAL_SMARTCARD_TxCpltCallback*
- *HAL_SMARTCARD_RxCpltCallback*
- *HAL_SMARTCARD_ErrorCallback*
- *HAL_SMARTCARD_AbortCpltCallback*
- *HAL_SMARTCARD_AbortTransmitCpltCallback*
- *HAL_SMARTCARD_AbortReceiveCpltCallback*

52.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard handle and also return Peripheral Errors occurred during communication process

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- *HAL_SMARTCARD_GetState*
- *HAL_SMARTCARD_GetError*

52.2.6 Detailed description of functions

HAL_SMARTCARD_Init

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_DeInit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD peripheral.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_MspInit

Function name

void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_MspDeInit

Function name

void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_Transmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_SMARTCARD_Receive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_SMARTCARD_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_TDR register is empty, i.e one interrupt per data to transmit.
- When FIFO mode is enabled, USART interrupt is generated whenever TXFIFO threshold reached. In that case the interrupt rate depends on TXFIFO threshold configuration.
- This function sets the hsmartcard->TxIsr function pointer according to the FIFO mode (data transmission processing depends on FIFO mode).

HAL_SMARTCARD_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_RDR register can be read, i.e one interrupt per data to receive.
- When FIFO mode is enabled, USART interrupt is generated whenever RXFIFO threshold reached. In that case the interrupt rate depends on RXFIFO threshold configuration.
- This function sets the hsmartcard->RxIsr function pointer according to the FIFO mode (data reception processing depends on FIFO mode).

HAL_SMARTCARD_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_SMARTCARD_Abort

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name

void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Handle SMARTCARD interrupt requests.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_TxCpltCallback

Function name

void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Tx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RxCpltCallback

Function name

void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Rx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_ErrorCallback

Function name

void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD error callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name

void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name

void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name

void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Receive Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_GetState

Function name

HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle state.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle state

HAL_SMARTCARD_GetError

Function name

uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle error code.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle Error Code

52.3 SMARTCARD Firmware driver defines

The following section lists the various define and macros of the module.

52.3.1 SMARTCARD

SMARTCARD

SMARTCARD Clock Prescaler

SMARTCARD_PRESCALER_DIV1

fclk_pres = fclk

SMARTCARD_PRESCALER_DIV2

fclk_pres = fclk/2

SMARTCARD_PRESCALER_DIV4

fclk_pres = fclk/4

SMARTCARD_PRESCALER_DIV6

fclk_pres = fclk/6

SMARTCARD_PRESCALER_DIV8

fclk_pres = fclk/8

SMARTCARD_PRESCALER_DIV10

fclk_pres = fclk/10

SMARTCARD_PRESCALER_DIV12

fclk_pres = fclk/12

SMARTCARD_PRESCALER_DIV16

fclk_pres = fclk/16

SMARTCARD_PRESCALER_DIV32

fclk_pres = fclk/32

SMARTCARD_PRESCALER_DIV64

fclk_pres = fclk/64

SMARTCARD_PRESCALER_DIV128

fclk_pres = fclk/128

SMARTCARD_PRESCALER_DIV256

fclk_pres = fclk/256

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD frame phase on first clock transition

SMARTCARD_PHASE_2EDGE

SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD frame low polarity

SMARTCARD_POLARITY_HIGH

SMARTCARD frame high polarity

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

SMARTCARD Error Code Definition

HAL_SMARTCARD_ERROR_NONE

No error

HAL_SMARTCARD_ERROR_PE

Parity error

HAL_SMARTCARD_ERROR_NE

Noise error

HAL_SMARTCARD_ERROR_FE

frame error

HAL_SMARTCARD_ERROR_ORE

Overrun error

HAL_SMARTCARD_ERROR_DMA

DMA transfer error

HAL_SMARTCARD_ERROR_RTO

Receiver TimeOut error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle states.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard Data registers.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified SMARTCARD pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear flag

Return value:

- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

- Check whether the specified Smartcard flag is set or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag (when flag available)
  - SMARTCARD_FLAG_REACK Receive enable acknowledge flag
  - SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
  - SMARTCARD_FLAG_BUSY Busy flag
  - SMARTCARD_FLAG_EOBF End of block flag
  - SMARTCARD_FLAG_RTOF Receiver timeout flag
  - SMARTCARD_FLAG_TXE Transmit data register empty flag
  - SMARTCARD_FLAG_TC Transmission complete flag
  - SMARTCARD_FLAG_RXNE Receive data register not empty flag
  - SMARTCARD_FLAG_IDLE Idle line detection flag
  - SMARTCARD_FLAG_ORE Overrun error flag
  - SMARTCARD_FLAG_NE Noise error flag
  - SMARTCARD_FLAG_FE Framing error flag
  - SMARTCARD_FLAG_PE Parity error flag
  - SMARTCARD_FLAG_TXFNF TXFIFO not full flag
  - SMARTCARD_FLAG_RXFNE RXFIFO not empty flag
  - SMARTCARD_FLAG_TXFE TXFIFO Empty flag
  - SMARTCARD_FLAG_RXFF RXFIFO Full flag
  - SMARTCARD_FLAG_RXFT SMARTCARD RXFIFO threshold flag
  - SMARTCARD_FLAG_TXFT SMARTCARD TXFIFO threshold flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear Flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK

SMARTCARD interruptions flags mask

SMARTCARD_CR_MASK

SMARTCARD control register mask

SMARTCARD_CR_POS

SMARTCARD control register position

SMARTCARD_ISR_MASK

SMARTCARD ISR register mask

SMARTCARD_ISR_POS

SMARTCARD ISR register position

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_ENABLE

SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX

SMARTCARD RX mode

SMARTCARD_MODE_TX

SMARTCARD TX mode

SMARTCARD_MODE_TX_RX

SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_DISABLE

SMARTCARD NACK transmission disabled

SMARTCARD_NACK_ENABLE

SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE

SMARTCARD frame one-bit sample disabled

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD frame even parity

SMARTCARD_PARITY_ODD

SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST

Receive data flush request

SMARTCARD_TXDATA_FLUSH_REQUEST

Transmit data flush request

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

SMARTCARD State Code Definition

HAL_SMARTCARD_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_SMARTCARD_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_SMARTCARD_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_SMARTCARD_STATE_ERROR

Error Value is allowed for gState only

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD frame with 0.5 stop bit

SMARTCARD_STOPBITS_1_5

SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE

SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE

SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

SMARTCARD frame length

53 HAL SMARTCARD Extension Driver

53.1 SMARTCARDEx Firmware driver API description

The following section lists the various functions of the SMARTCARDEx library.

53.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
2. FIFO mode enabling/disabling and RX/TX FIFO threshold programming.

Note: When SMARTCARD operates in FIFO mode, FIFO mode must be enabled prior starting RX/TX transfers. Also RX/TX FIFO thresholds must be configured prior starting RX/TX transfers.

53.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEx_DisableReceiverTimeOut() API disables the receiver timeout feature

This section contains the following APIs:

- ***HAL_SMARTCARDEx_BlockLength_Config***
- ***HAL_SMARTCARDEx_TimeOut_Config***
- ***HAL_SMARTCARDEx_EnableReceiverTimeOut***
- ***HAL_SMARTCARDEx_DisableReceiverTimeOut***

53.1.3 IO operation functions

This subsection provides a set of FIFO mode related callback functions.

1. TX/RX Fifos Callbacks:
   - HAL_SMARTCARDEx_RxFifoFullCallback()
   - HAL_SMARTCARDEx_TxFifoEmptyCallback()

This section contains the following APIs:

- ***HAL_SMARTCARDEx_RxFifoFullCallback***
- ***HAL_SMARTCARDEx_TxFifoEmptyCallback***

53.1.4 Peripheral FIFO Control functions

This subsection provides a set of functions allowing to control the SMARTCARD FIFO feature.

- HAL_SMARTCARDEx_EnableFifoMode() API enables the FIFO mode
- HAL_SMARTCARDEx_DisableFifoMode() API disables the FIFO mode
- HAL_SMARTCARDEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_SMARTCARDEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_SMARTCARDEx_EnableFifoMode***
- ***HAL_SMARTCARDEx_DisableFifoMode***
- ***HAL_SMARTCARDEx_SetTxFifoThreshold***
- ***HAL_SMARTCARDEx_SetRxFifoThreshold***

53.1.5 Detailed description of functions

HAL_SMARTCARDEx_BlockLength_Config

Function name

void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)

Function description

Update on the fly the SMARTCARD block length in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **BlockLength:** SMARTCARD block length (8-bit long at most)

Return values

- **None:**

HAL_SMARTCARDEx_TimeOut_Config

Function name

void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.

Return values

- **None:**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_RxFifoFullCallback

Function name

void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD RX Fifo full callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name

void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD TX Fifo empty callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - SMARTCARD_TXFIFO_THRESHOLD_1_8
  - SMARTCARD_TXFIFO_THRESHOLD_1_4
  - SMARTCARD_TXFIFO_THRESHOLD_1_2
  - SMARTCARD_TXFIFO_THRESHOLD_3_4
  - SMARTCARD_TXFIFO_THRESHOLD_7_8
  - SMARTCARD_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - SMARTCARD_RXFIFO_THRESHOLD_1_8
  - SMARTCARD_RXFIFO_THRESHOLD_1_4
  - SMARTCARD_RXFIFO_THRESHOLD_1_2
  - SMARTCARD_RXFIFO_THRESHOLD_3_4
  - SMARTCARD_RXFIFO_THRESHOLD_7_8
  - SMARTCARD_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

53.2 SMARTCARDEx Firmware driver defines

The following section lists the various define and macros of the module.

53.2.1 SMARTCARDEx

SMARTCARDEx

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

No advanced feature initialization

SMARTCARD_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

SMARTCARD_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

SMARTCARD_ADVFEATURE_SWAP_INIT

TX/RX pins swap

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

SMARTCARD_ADVFEATURE_TXCOMPLETION

TX completion indication before of after guard time

SMARTCARD FIFO mode

SMARTCARD_FIFOMODE_DISABLE

FIFO mode disable

SMARTCARD_FIFOMODE_ENABLE

FIFO mode enable

SMARTCARD Flags

SMARTCARD_FLAG_TCBGT

SMARTCARD transmission complete before guard time completion

SMARTCARD_FLAG_REACK

SMARTCARD receive enable acknowledge flag

SMARTCARD_FLAG_TEACK

SMARTCARD transmit enable acknowledge flag

SMARTCARD_FLAG_BUSY

SMARTCARD busy flag

SMARTCARD_FLAG_EOBF

SMARTCARD end of block flag

SMARTCARD_FLAG_RTOF

SMARTCARD receiver timeout flag

SMARTCARD_FLAG_TXE

SMARTCARD transmit data register empty

SMARTCARD_FLAG_TXFNF

SMARTCARD TXFIFO not full

SMARTCARD_FLAG_TC

SMARTCARD transmission complete

SMARTCARD_FLAG_RXNE

SMARTCARD read data register not empty

SMARTCARD_FLAG_RXFNE

SMARTCARD RXFIFO not empty

SMARTCARD_FLAG_IDLE

SMARTCARD idle line detection

SMARTCARD_FLAG_ORE

SMARTCARD overrun error

SMARTCARD_FLAG_NE

SMARTCARD noise error

SMARTCARD_FLAG_FE

SMARTCARD frame error

SMARTCARD_FLAG_PE

SMARTCARD parity error

SMARTCARD_FLAG_TXFE

SMARTCARD TXFIFO Empty flag

SMARTCARD_FLAG_RXFF

SMARTCARD RXFIFO Full flag

SMARTCARD_FLAG_RXFT

SMARTCARD RXFIFO threshold flag

SMARTCARD_FLAG_TXFT

SMARTCARD TXFIFO threshold flag

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE

SMARTCARD parity error interruption

SMARTCARD_IT_TXE

SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TXFNF

SMARTCARD TX FIFO not full interruption

SMARTCARD_IT_TC

SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE

SMARTCARD read data register not empty interruption

SMARTCARD_IT_RXFNE

SMARTCARD RXFIFO not empty interruption

SMARTCARD_IT_IDLE

SMARTCARD idle line detection interruption

SMARTCARD_IT_ERR

SMARTCARD error interruption

SMARTCARD_IT_ORE

SMARTCARD overrun error interruption

SMARTCARD_IT_NE

SMARTCARD noise error interruption

SMARTCARD_IT_FE

SMARTCARD frame error interruption

SMARTCARD_IT_EOB

SMARTCARD end of block interruption

SMARTCARD_IT_RTO

SMARTCARD receiver timeout interruption

SMARTCARD_IT_TCBGT

SMARTCARD transmission complete before guard time completion interruption

SMARTCARD_IT_RXFF

SMARTCARD RXFIFO full interruption

SMARTCARD_IT_TXFE

SMARTCARD TXFIFO empty interruption

SMARTCARD_IT_RXFT

SMARTCARD RXFIFO threshold reached interruption

SMARTCARD_IT_TXFT

SMARTCARD TXFIFO threshold reached interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF

SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF

SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF

SMARTCARD noise error detected clear flag

SMARTCARD_CLEAR_OREF

SMARTCARD overrun error clear flag

SMARTCARD_CLEAR_IDLEF

SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TXFECF

TXFIFO empty Clear Flag

SMARTCARD_CLEAR_TCF

SMARTCARD transmission complete clear flag

SMARTCARD_CLEAR_TCBGTF

SMARTCARD transmission complete before guard time completion clear flag

SMARTCARD_CLEAR_RTOF

SMARTCARD receiver time out clear flag

SMARTCARD_CLEAR_EOBF

SMARTCARD end of block clear flag

SMARTCARD RXFIFO threshold level

SMARTCARD_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

SMARTCARD_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT

SMARTCARD transmission complete before guard time

SMARTCARD_TC

SMARTCARD transmission complete (flag raised when guard time has elapsed)

SMARTCARD TXFIFO threshold level

SMARTCARD_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

SMARTCARD_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

54 HAL SMBUS Generic Driver

54.1 SMBUS Firmware driver registers structures

54.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32g4xx_hal_smbus.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***

  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***

  Specifies if Analog Filter is enable or not. This parameter can be a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***

  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***

  Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***

  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***

  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***

  Specifies the acknoledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***

  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.
- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***

  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***
- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***

  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***

- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

54.1.2 SMBUS_HandleTypeDef

SMBUS_HandleTypeDef is defined in the stm32g4xx_hal_smbus.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***I2C_TypeDef* SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address
- ***SMBUS_InitTypeDef SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters
- ***uint8_t* SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer
- ***uint16_t SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size
- ***__IO uint16_t SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter
- ***__IO uint32_t SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options
- ***__IO uint32_t SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state
- ***HAL_LockTypeDef SMBUS_HandleTypeDef::Lock***
  SMBUS locking object
- ***__IO uint32_t SMBUS_HandleTypeDef::State***
  SMBUS communication state
- ***__IO uint32_t SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code

54.2 SMBUS Firmware driver API description

The following section lists the various functions of the SMBUS library.

54.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;

2. Initialize the SMBUS low level resources by implementing the @ref HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      - Enable the clock for the SMBUS GPIOs
      - Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the SMBUSx interrupt priority
      - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the @ref HAL_SMBUS_Init() API:
   – These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized @ref HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function @ref HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using @ref HAL_SMBUS_Master_Transmit_IT()
  – At transmission end of transfer @ref HAL_SMBUS_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using @ref HAL_SMBUS_Master_Receive_IT()
  – At reception end of transfer @ref HAL_SMBUS_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using @ref HAL_SMBUS_Master_Abort_IT()
  – The associated previous transfer callback is called at the end of abort process
  – mean @ref HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
  – mean @ref HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive
- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using @ref HAL_SMBUS_EnableListen_IT() @ref HAL_SMBUS_DisableListen_IT()
  – When address slave/device SMBUS match, @ref HAL_SMBUS_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  – At Listen mode end @ref HAL_SMBUS_ListenCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using @ref HAL_SMBUS_Slave_Transmit_IT()
  – At transmission end of transfer @ref HAL_SMBUS_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using @ref HAL_SMBUS_Slave_Receive_IT()
  – At reception end of transfer @ref HAL_SMBUS_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using @ref HAL_SMBUS_EnableAlert_IT() @ref HAL_SMBUS_DisableAlert_IT()
  – When SMBUS Alert is generated @ref HAL_SMBUS_ErrorCallback() is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function @ref HAL_SMBUS_GetError()
- Get HAL state machine or error values using @ref HAL_SMBUS_GetState() or @ref HAL_SMBUS_GetError()

- In case of transfer Error, @ref HAL_SMBUS_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_SMBUS_ErrorCallback() to check the Error Code using function @ref HAL_SMBUS_GetError()

SMBUS HAL driver macros list

Below the list of most used macros in SMBUS HAL driver.

- @ref __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- @ref __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- @ref __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- @ref __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- @ref __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- @ref __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

Callback registration

The compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_SMBUS_RegisterCallback() or @ref HAL_SMBUS_RegisterAddrCallback() to register an interrupt callback.

Function @ref HAL_SMBUS_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : @ref HAL_SMBUS_RegisterAddrCallback.

Use function @ref HAL_SMBUS_UnRegisterCallback to reset a callback to the default weak function. @ref HAL_SMBUS_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : @ref HAL_SMBUS_UnRegisterAddrCallback.

By default, after the @ref HAL_SMBUS_Init() and when the state is @ref HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples @ref HAL_SMBUS_MasterTxCpltCallback(), @ref HAL_SMBUS_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_SMBUS_Init()/ @ref HAL_SMBUS_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the @ref HAL_SMBUS_Init()/ @ref HAL_SMBUS_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in @ref HAL_I2C_STATE_READY or @ref HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_SMBUS_RegisterCallback() before calling @ref HAL_SMBUS_DeInit() or @ref HAL_SMBUS_Init() function.

When the compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the SMBUS HAL driver header file for more useful macros

54.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.
- Enable/Disable Analog/Digital filters with HAL_SMBUS_ConfigAnalogFilter() and HAL_SMBUS_ConfigDigitalFilter().

This section contains the following APIs:

- ***HAL_SMBUS_Init***
- ***HAL_SMBUS_DeInit***
- ***HAL_SMBUS_MspInit***
- ***HAL_SMBUS_MspDeInit***
- ***HAL_SMBUS_ConfigAnalogFilter***
- ***HAL_SMBUS_ConfigDigitalFilter***

54.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is :
   - HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
   - Non-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. Non-Blocking mode functions with Interrupt are :
   - HAL_SMBUS_Master_Transmit_IT()
   - HAL_SMBUS_Master_Receive_IT()
   - HAL_SMBUS_Slave_Transmit_IT()
   - HAL_SMBUS_Slave_Receive_IT()
   - HAL_SMBUS_EnableListen_IT() or alias HAL_SMBUS_EnableListen_IT()
   - HAL_SMBUS_DisableListen_IT()
   - HAL_SMBUS_EnableAlert_IT()
   - HAL_SMBUS_DisableAlert_IT()

4. A set of Transfer Complete Callbacks are provided in non-Blocking mode:
   - HAL_SMBUS_MasterTxCpltCallback()
   - HAL_SMBUS_MasterRxCpltCallback()
   - HAL_SMBUS_SlaveTxCpltCallback()
   - HAL_SMBUS_SlaveRxCpltCallback()
   - HAL_SMBUS_AddrCallback()
   - HAL_SMBUS_ListenCpltCallback()
   - HAL_SMBUS_ErrorCallback()

This section contains the following APIs:

- ***HAL_SMBUS_Master_Transmit_IT***
- ***HAL_SMBUS_Master_Receive_IT***
- ***HAL_SMBUS_Master_Abort_IT***
- ***HAL_SMBUS_Slave_Transmit_IT***
- ***HAL_SMBUS_Slave_Receive_IT***
- ***HAL_SMBUS_EnableListen_IT***
- ***HAL_SMBUS_DisableListen_IT***
- ***HAL_SMBUS_EnableAlert_IT***
- ***HAL_SMBUS_DisableAlert_IT***
- ***HAL_SMBUS_IsDeviceReady***

54.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_SMBUS_GetState***
- ***HAL_SMBUS_GetError***

54.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name

HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DeInit

Function name

HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS peripheral.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_MspInit

Function name

void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MspDeInit

Function name

void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigAnalogFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t AnalogFilter)

Function description

Configure Analog noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **AnalogFilter:** This parameter can be one of the following values:
  - SMBUS_ANALOGFILTER_ENABLE
  - SMBUS_ANALOGFILTER_DISABLE

Return values

- **HAL:** status

HAL_SMBUS_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigDigitalFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t DigitalFilter)

Function description

Configure Digital noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_SMBUS_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Check if target device is ready for communication.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SMBUS_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)

Function description

Abort a master/host SMBUS process communication with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

Notes

- This abort can be called only if state is ready

HAL_SMBUS_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name

void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS event interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ER_IRQHandler

Function name

void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS error interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterTxCpltCallback

Function name

void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterRxCpltCallback

Function name

void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveTxCpltCallback

Function name

void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveRxCpltCallback

Function name

void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_AddrCallback

Function name

void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **TransferDirection:** Master request Transfer Direction (Write/Read)
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_SMBUS_ListenCpltCallback

Function name

void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Listen Complete callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ErrorCallback

Function name

void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

SMBUS error callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_GetState

Function name

uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS handle state.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** state

HAL_SMBUS_GetError

Function name

uint32_t HAL_SMBUS_GetError (SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS error code.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **SMBUS:** Error Code

54.3 SMBUS Firmware driver defines

The following section lists the various define and macros of the module.

54.3.1 SMBUS

SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE

No error

HAL_SMBUS_ERROR_BERR

BERR error

HAL_SMBUS_ERROR_ARLO

ARLO error

HAL_SMBUS_ERROR_ACKF

ACKF error

HAL_SMBUS_ERROR_OVR

OVR error

HAL_SMBUS_ERROR_HALTIMEOUT

Timeout error

HAL_SMBUS_ERROR_BUSTIMEOUT

Bus Timeout error

HAL_SMBUS_ERROR_ALERT

Alert error

HAL_SMBUS_ERROR_PECERR

PEC error

HAL_SMBUS_ERROR_INVALID_PARAM

Invalid Parameters error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (SET or RESET).

SMBUS_FLAG_MASK

Description:

- Check whether the specified SMBUS flag is set or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_TXIS Transmit interrupt status
  - SMBUS_FLAG_RXNE Receive data register not empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_TC Transfer complete (master mode)
  - SMBUS_FLAG_TCR Transfer complete reload
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert
  - SMBUS_FLAG_BUSY Bus busy
  - SMBUS_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_GENERATE_NACK

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_FRAME_WITH_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

55 HAL SPI Generic Driver

55.1 SPI Firmware driver registers structures

55.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32g4xx_hal_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.
- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***

- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 1 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size This parameter can be a value of ***SPI_CRC_length***
- ***uint32_t SPI_InitTypeDef::NSSPMode***
  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode*** This mode is activated by the NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored)..

55.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32g4xx_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint32_t CRCSize***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter
- ***uint32_t __SPI_HandleTypeDef::CRCSize***
  SPI CRC size used for the transfer

- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code

55.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

55.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      - Enable the clock for the SPI GPIOs
      - Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process
      - Configure the SPIx interrupt priority
      - Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
      - Enable the DMAx clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx Stream/Channel
      - Associate the initialized hdma_tx(or _rx) handle to the hspi DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
   a. Master 2Lines RxOnly
   b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

Master Receive mode restriction:

1. In Master unidirectional receive-only mode (MSTR =1, BIDIMODE=0, RXONLY=1) or bidirectional receive mode (MSTR=1, BIDIMODE=1, BIDIOE=0), to ensure that the SPI does not initiate a new transfer the following procedure has to be respected:
   a. HAL_SPI_DeInit()
   b. HAL_SPI_Init()

Callback registration:

1. The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1U allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks:
   – TxCpltCallback : SPI Tx Completed callback
   – RxCpltCallback : SPI Rx Completed callback
   – TxRxCpltCallback : SPI TxRx Completed callback
   – TxHalfCpltCallback : SPI Tx Half Completed callback
   – RxHalfCpltCallback : SPI Rx Half Completed callback
   – TxRxHalfCpltCallback : SPI TxRx Half Completed callback
   – ErrorCallback : SPI Error callback
   – AbortCpltCallback : SPI Abort callback
   – MspInitCallback : SPI Msp Init callback
   – MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   – TxCpltCallback : SPI Tx Completed callback
   – RxCpltCallback : SPI Rx Completed callback
   – TxRxCpltCallback : SPI TxRx Completed callback
   – TxHalfCpltCallback : SPI Tx Half Completed callback
   – RxHalfCpltCallback : SPI Rx Half Completed callback
   – TxRxHalfCpltCallback : SPI TxRx Half Completed callback
   – ErrorCallback : SPI Error callback
   – AbortCpltCallback : SPI Abort callback
   – MspInitCallback : SPI Msp Init callback
   – MspDeInitCallback : SPI Msp DeInit callback

By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function.

When the compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI Modes, the following table resume the max SPI frequency reached with data size 8bits/16bits, according to frequency of the APBx Peripheral Clock (fPCLK) used by the SPI instance.

55.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- ***HAL_SPI_Init***
- ***HAL_SPI_DeInit***
- ***HAL_SPI_MspInit***
- ***HAL_SPI_MspDeInit***

55.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPI data transfers.

The SPI supports master and slave mode :

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback()user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- ***HAL_SPI_Transmit***
- ***HAL_SPI_Receive***
- ***HAL_SPI_TransmitReceive***
- ***HAL_SPI_Transmit_IT***
- ***HAL_SPI_Receive_IT***
- ***HAL_SPI_TransmitReceive_IT***
- ***HAL_SPI_Transmit_DMA***
- ***HAL_SPI_Receive_DMA***
- ***HAL_SPI_TransmitReceive_DMA***
- ***HAL_SPI_Abort***
- ***HAL_SPI_Abort_IT***
- ***HAL_SPI_DMAPause***
- ***HAL_SPI_DMAResume***
- ***HAL_SPI_DMAStop***

- *HAL_SPI_IRQHandler*
- *HAL_SPI_TxCpltCallback*
- *HAL_SPI_RxCpltCallback*
- *HAL_SPI_TxRxCpltCallback*
- *HAL_SPI_TxHalfCpltCallback*
- *HAL_SPI_RxHalfCpltCallback*
- *HAL_SPI_TxRxHalfCpltCallback*
- *HAL_SPI_ErrorCallback*
- *HAL_SPI_AbortCpltCallback*

55.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- HAL_SPI_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL_SPI_GetError() check in run-time Errors occurring during communication

This section contains the following APIs:

- *HAL_SPI_GetState*
- *HAL_SPI_GetError*

55.2.5 Detailed description of functions

HAL_SPI_Init

Function name

HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_DeInit

Function name

HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI peripheral.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_MspInit

Function name

void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_MspDeInit

Function name

void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_Transmit

Function name

HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Receive

Function name

HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_TransmitReceive

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Transmit and Receive an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received

Return values

- **HAL:** status

HAL_SPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

Notes

- In case of MASTER mode and SPI_DIRECTION_2LINES direction, hdmatx shall be defined.
- When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name

HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)

Function description

Pause the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPI_DMAResume

Function name

HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)

Function description

Resume the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPI_DMAStop

Function name

HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)

Function description

Stop the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPI_Abort

Function name

HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (blocking mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode. This procedure performs following operations : Disable SPI Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (Interrupt mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode. This procedure performs following operations : Disable SPI Interrupts (depending of transfer direction)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SPI_IRQHandler

Function name

void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)

Function description

Handle SPI interrupt request.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **None:**

HAL_SPI_TxCpltCallback

Function name

void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxCpltCallback

Function name

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxCpltCallback

Function name

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxHalfCpltCallback

Function name

void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxHalfCpltCallback

Function name

void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name

void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Half Transfer callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_ErrorCallback

Function name

void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)

Function description

SPI error callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_AbortCpltCallback

Function name

void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Abort Complete callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_GetState

Function name

HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)

Function description

Return the SPI handle state.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** state

HAL_SPI_GetError

Function name

uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)

Function description

Return the SPI error code.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** error code in bitmap format

55.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

55.3.1 SPI

SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_16BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Code

HAL_SPI_ERROR_NONE

No error

HAL_SPI_ERROR_MODF

MODF error

HAL_SPI_ERROR_CRC

CRC error

HAL_SPI_ERROR_OVR

OVR error

HAL_SPI_ERROR_FRE

FRE error

HAL_SPI_ERROR_DMA

DMA transfer error

HAL_SPI_ERROR_FLAG

Error on RXNE/TXE/BSY/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT

Error during SPI Abort procedure

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPI_GET_FLAG

Description:

- Check whether the specified SPI flag is set or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXNE: Receive buffer not empty flag
  - SPI_FLAG_TXE: Transmit buffer empty flag
  - SPI_FLAG_CRCERR: CRC error flag
  - SPI_FLAG_MODF: Mode fault flag
  - SPI_FLAG_OVR: Overrun flag
  - SPI_FLAG_BSY: Busy flag
  - SPI_FLAG_FRE: Frame format error flag
  - SPI_FLAG_FTLVL: SPI fifo transmission level
  - SPI_FLAG_FRLVL: SPI fifo reception level

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI FIFO Reception Threshold

SPI_RXFIFO_THRESHOLD

SPI_RXFIFO_THRESHOLD_QF

SPI_RXFIFO_THRESHOLD_HF

SPI Flags Definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_BSY

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_FRE

SPI_FLAG_FTLVL

SPI_FLAG_FRLVL

SPI_FLAG_MASK

SPI Interrupt Definition

SPI_IT_TXE

SPI_IT_RXNE

SPI_IT_ERR

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_ENABLE

SPI_NSS_PULSE_DISABLE

SPI Reception FIFO Status Level

SPI_FRLVL_EMPTY

SPI_FRLVL_QUARTER_FULL

SPI_FRLVL_HALF_FULL

SPI_FRLVL_FULL

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Transmission FIFO Status Level

SPI_FTLVL_EMPTY

SPI_FTLVL_QUARTER_FULL

SPI_FTLVL_HALF_FULL

SPI_FTLVL_FULL

56 HAL SPI Extension Driver

56.1 SPIEx Firmware driver API description

The following section lists the various functions of the SPIEx library.

56.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. Rx data flush function:
   – HAL_SPIEx_FlushRxFifo()

This section contains the following APIs:

- ***HAL_SPIEx_FlushRxFifo***

56.1.2 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (SPI_HandleTypeDef * hspi)

Function description

Flush the RX fifo.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

57 HAL SRAM Generic Driver

57.1 SRAM Firmware driver registers structures

57.1.1 SRAM_HandleTypeDef

SRAM_HandleTypeDef is defined in the stm32g4xx_hal_sram.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***

Field Documentation

- ***FMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* SRAM_HandleTypeDef::hdma***
  Pointer DMA handler

57.2 SRAM Firmware driver API description

The following section lists the various functions of the SRAM library.

57.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.

3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
   b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
   c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   – HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

Callback registration

The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_SRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function @ref HAL_SRAM_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_SRAM_Init and if the state is HAL_SRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_SRAM_Init and @ref HAL_SRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_SRAM_Init and @ref HAL_SRAM_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_SRAM_RegisterCallback before calling @ref HAL_SRAM_DeInit or @ref HAL_SRAM_Init function. When The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

57.2.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- ***HAL_SRAM_Init***
- ***HAL_SRAM_DeInit***
- ***HAL_SRAM_MspInit***
- ***HAL_SRAM_MspDeInit***
- ***HAL_SRAM_DMA_XferCpltCallback***
- ***HAL_SRAM_DMA_XferErrorCallback***

57.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- *HAL_SRAM_Read_8b*
- *HAL_SRAM_Write_8b*
- *HAL_SRAM_Read_16b*
- *HAL_SRAM_Write_16b*
- *HAL_SRAM_Read_32b*
- *HAL_SRAM_Write_32b*
- *HAL_SRAM_Read_DMA*
- *HAL_SRAM_Write_DMA*
- *HAL_SRAM_DMA_XferCpltCallback*
- *HAL_SRAM_DMA_XferErrorCallback*

57.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- *HAL_SRAM_WriteOperation_Enable*
- *HAL_SRAM_WriteOperation_Disable*

57.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

This section contains the following APIs:

- *HAL_SRAM_GetState*

57.2.6 Detailed description of functions

HAL_SRAM_Init

Function name

HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Performs the SRAM device initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **Timing:** Pointer to SRAM control timing structure
- **ExtTiming:** Pointer to SRAM extended mode timing structure

Return values

- **HAL:** status

HAL_SRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)

Function description

Performs the SRAM device De-initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_MspInit

Function name

void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP Init.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_MspDeInit

Function name

void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP DeInit.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 8-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 8-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_DMA_XferCpltCallback

Function name

void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name

void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete error callback.

Parameters

- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)

Function description

Enables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)

Function description

Disables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_GetState

Function name

HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)

Function description

Returns the SRAM controller state.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** state

57.3 SRAM Firmware driver defines

The following section lists the various define and macros of the module.

57.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:

- Reset SRAM handle state.

Parameters:

- __HANDLE__: SRAM handle

Return value:

- None

58 HAL TIM Generic Driver

58.1 TIM Firmware driver registers structures

58.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***

  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF Macro **__HAL_TIM_CALC_PSC()** can be used to calculate prescaler value
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***

  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***

  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PERIOD()**, **__HAL_TIM_CALC_PERIOD_DITHER()**, **__HAL_TIM_CALC_PERIOD_BY_DELAY()**, **__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY()** can be used to calculate Period value
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***

  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***

  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***

  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

58.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***

  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse***

  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***

  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***

  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***

  Note:

  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***

  Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***

  Note:

  – This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***

  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***

  Note:

  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***

  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***

  Note:

  – This parameter is valid only for timer instances supporting break feature.

58.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***

  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***

  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***

  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***

- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

58.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

58.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1Selection***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2Selection***

- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t TIM_Encoder_InitTypeDef::EncoderMode***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Mode***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

58.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
  TIM clock sources This parameter can be a value of ***TIM_Clock_Source***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
  TIM clock polarity This parameter can be a value of ***TIM_Clock_Polarity***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
  TIM clock prescaler This parameter can be a value of ***TIM_Clock_Prescaler***
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
  TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

58.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***

- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter must be 0: When OCRef clear feature is used with ETR source, ETR prescaler must be off
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

58.1.8 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM_Master_Mode_Selection_2***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***
  Note:
  – When the Master/slave mode is enabled, the effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is not mandatory in case of timer synchronization mode.

58.1.9 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***

- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

58.1.10 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***
- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the break input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input. This parameter can be a value of ***TIM_Break_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State This parameter can be a value of ***TIM_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity This parameter can be a value of ***TIM_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input. This parameter can be a value of ***TIM_Break2_Input_AF_Mode***

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

58.1.11 TIM_HandleTypeDef

TIM_HandleTypeDef is defined in the stm32g4xx_hal_tim.h

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelState***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelNState***
- ***__IO HAL_TIM_DMABurstStateTypeDef DMABurstState***

Field Documentation

- ***TIM_TypeDef* TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a ***DMA_Handle_index***
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
  TIM operation state
- ***__IO HAL_TIM_ChannelStateTypeDef TIM_HandleTypeDef::ChannelState[6]***
  TIM channel operation state
- ***__IO HAL_TIM_ChannelStateTypeDef TIM_HandleTypeDef::ChannelNState[4]***
  TIM complementary channel operation state
- ***__IO HAL_TIM_DMABurstStateTypeDef TIM_HandleTypeDef::DMABurstState***
  DMA burst operation state

58.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

58.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental encoder for positioning purposes

58.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   - Time Base : HAL_TIM_Base_MspInit()
   - Input Capture : HAL_TIM_IC_MspInit()
   - Output Compare : HAL_TIM_OC_MspInit()
   - PWM generation : HAL_TIM_PWM_MspInit()
   - One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   - Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      - Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
   - HAL_TIM_Base_Init: to use the Timer to generate a simple time base
   - HAL_TIM_OC_Init, HAL_TIM_OC_ConfigChannel and optionally HAL_TIMEx_OC_ConfigPulseOnCompare: to use the Timer to generate an Output Compare signal.
   - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
   - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
   - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
   - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
   - Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
   - Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
   - Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
   - PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
   - One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
   - Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

Callback registration

The compilation define USE_HAL_TIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_TIM_RegisterCallback() to register a callback. @ref HAL_TIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_TIM_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_TIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- Base_MspInitCallback : TIM Base Msp Init Callback.
- Base_MspDeInitCallback : TIM Base Msp DeInit Callback.
- IC_MspInitCallback : TIM IC Msp Init Callback.
- IC_MspDeInitCallback : TIM IC Msp DeInit Callback.
- OC_MspInitCallback : TIM OC Msp Init Callback.

- OC_MspDeInitCallback : TIM OC Msp DeInit Callback.
- PWM_MspInitCallback : TIM PWM Msp Init Callback.
- PWM_MspDeInitCallback : TIM PWM Msp DeInit Callback.
- OnePulse_MspInitCallback : TIM One Pulse Msp Init Callback.
- OnePulse_MspDeInitCallback : TIM One Pulse Msp DeInit Callback.
- Encoder_MspInitCallback : TIM Encoder Msp Init Callback.
- Encoder_MspDeInitCallback : TIM Encoder Msp DeInit Callback.
- HallSensor_MspInitCallback : TIM Hall Sensor Msp Init Callback.
- HallSensor_MspDeInitCallback : TIM Hall Sensor Msp DeInit Callback.
- PeriodElapsedCallback : TIM Period Elapsed Callback.
- PeriodElapsedHalfCpltCallback : TIM Period Elapsed half complete Callback.
- TriggerCallback : TIM Trigger Callback.
- TriggerHalfCpltCallback : TIM Trigger half complete Callback.
- IC_CaptureCallback : TIM Input Capture Callback.
- IC_CaptureHalfCpltCallback : TIM Input Capture half complete Callback.
- OC_DelayElapsedCallback : TIM Output Compare Delay Elapsed Callback.
- PWM_PulseFinishedCallback : TIM PWM Pulse Finished Callback.
- PWM_PulseFinishedHalfCpltCallback : TIM PWM Pulse Finished half complete Callback.
- ErrorCallback : TIM Error Callback.
- CommutationCallback : TIM Commutation Callback.
- CommutationHalfCpltCallback : TIM Commutation half complete Callback.
- BreakCallback : TIM Break Callback.
- Break2Callback : TIM Break2 Callback.
- EncoderIndexCallback : TIM Encoder Index Callback.
- DirectionChangeCallback : TIM Direction Change Callback
- IndexErrorCallback : TIM Index Error Callback.
- TransitionErrorCallback : TIM Transition Error Callback

By default, after the Init and when the state is HAL_TIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples @ref HAL_TIM_TriggerCallback(), @ref HAL_TIM_ErrorCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init / DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init / DeInit keep and use the user MspInit / MspDeInit callbacks(registered beforehand)

Callbacks can be registered / unregistered in HAL_TIM_STATE_READY state only. Exception done MspInit / MspDeInit that can be registered / unregistered in HAL_TIM_STATE_READY or HAL_TIM_STATE_RESET state, thus registered(user) MspInit / DeInit callbacks can be used during the Init / DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_TIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_TIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

58.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_Base_Init*
- *HAL_TIM_Base_DeInit*
- *HAL_TIM_Base_MspInit*
- *HAL_TIM_Base_MspDeInit*
- *HAL_TIM_Base_Start*
- *HAL_TIM_Base_Stop*
- *HAL_TIM_Base_Start_IT*
- *HAL_TIM_Base_Stop_IT*
- *HAL_TIM_Base_Start_DMA*
- *HAL_TIM_Base_Stop_DMA*

58.2.4 TIM Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the TIM Output Compare.
- Stop the TIM Output Compare.
- Start the TIM Output Compare and enable interrupt.
- Stop the TIM Output Compare and disable interrupt.
- Start the TIM Output Compare and enable DMA transfer.
- Stop the TIM Output Compare and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_OC_Init*
- *HAL_TIM_OC_DeInit*
- *HAL_TIM_OC_MspInit*
- *HAL_TIM_OC_MspDeInit*
- *HAL_TIM_OC_Start*
- *HAL_TIM_OC_Stop*
- *HAL_TIM_OC_Start_IT*
- *HAL_TIM_OC_Stop_IT*
- *HAL_TIM_OC_Start_DMA*
- *HAL_TIM_OC_Stop_DMA*

58.2.5 TIM PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the TIM PWM.
- Stop the TIM PWM.
- Start the TIM PWM and enable interrupt.
- Stop the TIM PWM and disable interrupt.
- Start the TIM PWM and enable DMA transfer.
- Stop the TIM PWM and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_PWM_Init*
- *HAL_TIM_PWM_DeInit*
- *HAL_TIM_PWM_MspInit*
- *HAL_TIM_PWM_MspDeInit*
- *HAL_TIM_PWM_Start*

- *HAL_TIM_PWM_Stop*
- *HAL_TIM_PWM_Start_IT*
- *HAL_TIM_PWM_Stop_IT*
- *HAL_TIM_PWM_Start_DMA*
- *HAL_TIM_PWM_Stop_DMA*

58.2.6 TIM Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the TIM Input Capture.
- Stop the TIM Input Capture.
- Start the TIM Input Capture and enable interrupt.
- Stop the TIM Input Capture and disable interrupt.
- Start the TIM Input Capture and enable DMA transfer.
- Stop the TIM Input Capture and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_IC_Init*
- *HAL_TIM_IC_DeInit*
- *HAL_TIM_IC_MspInit*
- *HAL_TIM_IC_MspDeInit*
- *HAL_TIM_IC_Start*
- *HAL_TIM_IC_Stop*
- *HAL_TIM_IC_Start_IT*
- *HAL_TIM_IC_Stop_IT*
- *HAL_TIM_IC_Start_DMA*
- *HAL_TIM_IC_Stop_DMA*

58.2.7 TIM One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the TIM One Pulse.
- Stop the TIM One Pulse.
- Start the TIM One Pulse and enable interrupt.
- Stop the TIM One Pulse and disable interrupt.
- Start the TIM One Pulse and enable DMA transfer.
- Stop the TIM One Pulse and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_OnePulse_Init*
- *HAL_TIM_OnePulse_DeInit*
- *HAL_TIM_OnePulse_MspInit*
- *HAL_TIM_OnePulse_MspDeInit*
- *HAL_TIM_OnePulse_Start*
- *HAL_TIM_OnePulse_Stop*
- *HAL_TIM_OnePulse_Start_IT*
- *HAL_TIM_OnePulse_Stop_IT*

58.2.8 TIM Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the TIM Encoder.
- Stop the TIM Encoder.
- Start the TIM Encoder and enable interrupt.
- Stop the TIM Encoder and disable interrupt.
- Start the TIM Encoder and enable DMA transfer.
- Stop the TIM Encoder and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_Encoder_Init***
- ***HAL_TIM_Encoder_DeInit***
- ***HAL_TIM_Encoder_MspInit***
- ***HAL_TIM_Encoder_MspDeInit***
- ***HAL_TIM_Encoder_Start***
- ***HAL_TIM_Encoder_Stop***
- ***HAL_TIM_Encoder_Start_IT***
- ***HAL_TIM_Encoder_Stop_IT***
- ***HAL_TIM_Encoder_Start_DMA***
- ***HAL_TIM_Encoder_Stop_DMA***

58.2.9 TIM Callbacks functions

This section provides TIM callback functions:

- TIM Period elapsed callback
- TIM Output Compare callback
- TIM Input capture callback
- TIM Trigger callback
- TIM Error callback
- TIM Index callback
- TIM Direction change callback
- TIM Index error callback
- TIM Transition error callback

This section contains the following APIs:

- ***HAL_TIM_PeriodElapsedCallback***
- ***HAL_TIM_PeriodElapsedHalfCpltCallback***
- ***HAL_TIM_OC_DelayElapsedCallback***
- ***HAL_TIM_IC_CaptureCallback***
- ***HAL_TIM_IC_CaptureHalfCpltCallback***
- ***HAL_TIM_PWM_PulseFinishedCallback***
- ***HAL_TIM_PWM_PulseFinishedHalfCpltCallback***
- ***HAL_TIM_TriggerCallback***
- ***HAL_TIM_TriggerHalfCpltCallback***
- ***HAL_TIM_ErrorCallback***

58.2.10 Detailed description of functions

HAL_TIM_Base_Init

Function name

HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Base_DeInit() before HAL_TIM_Base_Init()

HAL_TIM_Base_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Base peripheral.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_MspInit

Function name

void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_MspDeInit

Function name

void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_Start

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to peripheral.

Return values

- **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_OC_Init

Function name

HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OC_DeInit() before HAL_TIM_OC_Init()

HAL_TIM_OC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

HAL_TIM_OC_MspInit

Function name

void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_MspDeInit

Function name

void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_Start

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Init

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_PWM_DeInit() before HAL_TIM_PWM_Init()

HAL_TIM_PWM_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

HAL_TIM_PWM_MspInit

Function name

void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_MspDeInit

Function name

void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Init

Function name

HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_IC_DeInit() before HAL_TIM_IC_Init()

HAL_TIM_IC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

HAL_TIM_IC_MspInit

Function name

void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture MSP.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **None:**

HAL_TIM_IC_MspDeInit

Function name

void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Input Capture MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_IC_Start

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Init

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)

Function description

Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM One Pulse handle
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values:
  - TIM_OPMODE_SINGLE: Only one pulse will be generated.
  - TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OnePulse_DeInit() before HAL_TIM_OnePulse_Init()
- When the timer instance is initialized in One Pulse mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_OnePulse_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM One Pulse.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name

void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_MspDeInit

Function name

void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channels to be disable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Init

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)

Function description

Initializes the TIM Encoder Interface and initialize the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle
- **sConfig:** TIM Encoder Interface configuration structure

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Encoder_DeInit() before HAL_TIM_Encoder_Init()
- Encoder mode and External clock mode 2 are not compatible and must not be selected together Ex: A call for HAL_TIM_Encoder_Init will erase the settings of HAL_TIM_ConfigClockSource using TIM_CLOCKSOURCE_ETRMODE2 and vice versa
- When the timer instance is initialized in Encoder mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_Encoder_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Encoder interface.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** status

HAL_TIM_Encoder_MspInit

Function name

void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_MspDeInit

Function name

void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description

Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_IRQHandler

Function name

void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

Function description

This function handles TIM interrupts requests.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM Output Compare handle
- **sConfig:** TIM Output Compare configuration structure
- **Channel:** TIM Channels to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM PWM handle
- **sConfig:** TIM PWM configuration structure
- **Channel:** TIM Channels to be configured This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters

- **htim:** TIM IC handle
- **sConfig:** TIM Input Capture configuration structure
- **Channel:** TIM Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)

Function description

Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim:** TIM One Pulse handle
- **sConfig:** TIM One Pulse configuration structure
- **OutputChannel:** TIM output channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** TIM input Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- To output a waveform with a minimum delay user can enable the fast mode by calling the __HAL_TIM_ENABLE_OCxFAST macro. Then CCx output is forced in response to the edge detection on TIx input, without taking in account the comparison.

HAL_TIM_ConfigOCrefClear

Function name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function description

Configures the OCRef clear feature.

Parameters

- **htim:** TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **HAL:** status

HAL_TIM_ConfigClockSource

Function name

HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)

Function description

Configures the clock source to be used.

Parameters

- **htim:** TIM handle
- **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values

- **HAL:** status

HAL_TIM_ConfigTI1Input

Function name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function description

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters

- **htim:** TIM handle.
- **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
  - TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input
  - TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1, Reset + Trigger, Gated + Reset).

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro_IT

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode in interrupt mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1, Reset + Trigger, Gated + Reset).

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_OR
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiWriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiWriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer multiple Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_OR
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stops the TIM DMA Burst mode.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_OR
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_OR
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stop the DMA burst reading.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable.

Return values

- **HAL:** status

HAL_TIM_GenerateEvent

Function name

HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)

Function description

Generate a software event.

Parameters

- **htim:** TIM handle
- **EventSource:** specifies the event source. This parameter can be one of the following values:
  - TIM_EVENTSOURCE_UPDATE: Timer update Event source
  - TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  - TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source
  - TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
  - TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
  - TIM_EVENTSOURCE_COM: Timer COM event source
  - TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
  - TIM_EVENTSOURCE_BREAK: Timer Break event source
  - TIM_EVENTSOURCE_BREAK2: Timer Break2 event source

Return values

- **HAL:** status

Notes

- Basic timers can only generate an update event.
- TIM_EVENTSOURCE_COM is relevant only with advanced timer instances.
- TIM_EVENTSOURCE_BREAK and TIM_EVENTSOURCE_BREAK2 are relevant only for timer instances supporting break input(s).

HAL_TIM_ReadCapturedValue

Function name

uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Read the captured value from Capture Compare unit.

Parameters

- **htim:** TIM handle.
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PeriodElapsedHalfCpltCallback

Function name

void HAL_TIM_PeriodElapsedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name

void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Output Compare callback in non-blocking mode.

Parameters

- **htim:** TIM OC handle

Return values

- **None:**

HAL_TIM_IC_CaptureCallback

Function name

void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_IC_CaptureHalfCpltCallback

Function name

void HAL_TIM_IC_CaptureHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture half complete callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name

void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedHalfCpltCallback

Function name

void HAL_TIM_PWM_PulseFinishedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerCallback

Function name

void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerHalfCpltCallback

Function name

void HAL_TIM_TriggerHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_ErrorCallback

Function name

void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

Function description

Timer error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_Base_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Base handle state.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** state

HAL_TIM_OC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM OC handle state.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** state

HAL_TIM_PWM_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM PWM handle state.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** state

HAL_TIM_IC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Input Capture handle state.

Parameters

- **htim:** TIM IC handle

Return values

- **HAL:** state

HAL_TIM_OnePulse_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM One Pulse Mode handle state.

Parameters

- **htim:** TIM OPM handle

Return values

- **HAL:** state

HAL_TIM_Encoder_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** state

HAL_TIM_GetActiveChannel

Function name

HAL_TIM_ActiveChannel HAL_TIM_GetActiveChannel (TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM handle

Return values

- **Active:** channel

HAL_TIM_GetChannelState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIM_GetChannelState (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Return actual state of the TIM channel.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **TIM:** Channel state

HAL_TIM_DMABurstState

Function name

HAL_TIM_DMABurstStateTypeDef HAL_TIM_DMABurstState (TIM_HandleTypeDef * htim)

Function description

Return actual state of a DMA burst operation.

Parameters

- **htim:** TIM handle

Return values

- **DMA:** burst state

TIM_Base_SetConfig

Function name

void TIM_Base_SetConfig (TIM_TypeDef * TIMx, TIM_Base_InitTypeDef * Structure)

Function description

Time Base configuration.

Parameters

- **TIMx:** TIM peripheral
- **Structure:** TIM Base configuration structure

Return values

- **None:**

TIM_TI1_SetConfig

Function name

void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)

Function description

Configure the TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** The Input Polarity. This parameter can be one of the following values:
  - TIM_ICPOLARITY_RISING
  - TIM_ICPOLARITY_FALLING
  - TIM_ICPOLARITY_BOTHEDGE
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
  - TIM_ICSELECTION_DIRECTTI: TIM Input 1 is selected to be connected to IC1.
  - TIM_ICSELECTION_INDIRECTTI: TIM Input 1 is selected to be connected to IC2.
  - TIM_ICSELECTION_TRC: TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values

- **None:**

Notes

- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name

void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)

Function description

Timer Output Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The ouput configuration structure

Return values

- **None:**

TIM_ETR_SetConfig

Function name

void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function description

Configures the TIMx External Trigger (ETR).

Parameters

- **TIMx:** to select the TIM peripheral
- **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values:
  - TIM_ETRPRESCALER_DIV1: ETRP Prescaler OFF.
  - TIM_ETRPRESCALER_DIV2: ETRP frequency divided by 2.
  - TIM_ETRPRESCALER_DIV4: ETRP frequency divided by 4.
  - TIM_ETRPRESCALER_DIV8: ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values:
  - TIM_ETRPOLARITY_INVERTED: active low or falling edge active.
  - TIM_ETRPOLARITY_NONINVERTED: active high or rising edge active.
- **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values

- **None:**

TIM_DMADelayPulseHalfCplt

Function name

void TIM_DMADelayPulseHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Delay Pulse half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMAError

Function name

void TIM_DMAError (DMA_HandleTypeDef * hdma)

Function description

TIM DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureCplt

Function name

void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureHalfCplt

Function name

void TIM_DMACaptureHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

Enables or disables the TIM Capture Compare Channel x.

Parameters

- **TIMx:** to select the TIM peripheral
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_DISABLE.

Return values

- **None:**

58.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

58.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE

TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE

TIMx_ARR register is buffered

TIM Break2 Input Alternate Function Mode

TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

TIM Break input 2 Enable

TIM_BREAK2_DISABLE

Break input BRK2 is disabled

TIM_BREAK2_ENABLE

Break input BRK2 is enabled

TIM Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

Break input BRK2 is active low

TIM_BREAK2POLARITY_HIGH

Break input BRK2 is active high

TIM Break Input Alternate Function Mode

TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

TIM Break Input Enable

TIM_BREAK_ENABLE

Break input BRK is enabled

TIM_BREAK_DISABLE

Break input BRK is disabled

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

Break input BRK is active low

TIM_BREAKPOLARITY_HIGH

Break input BRK is active high

TIM Break System

TIM_BREAK_SYSTEM_ECC

Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17/20

TIM_BREAK_SYSTEM_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17/20 Break Input and also the PVDE and PLS bits of the Power Control Interface

TIM_BREAK_SYSTEM_SRAM_PARITY_ERROR

Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/8/15/16/17/20

TIM_BREAK_SYSTEM_LOCKUP

Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/8/15/16/17/20

TIM Channel

TIM_CHANNEL_1

Capture/compare channel 1 identifier

TIM_CHANNEL_2

Capture/compare channel 2 identifier

TIM_CHANNEL_3

Capture/compare channel 3 identifier

TIM_CHANNEL_4

Capture/compare channel 4 identifier

TIM_CHANNEL_5

Compare channel 5 identifier

TIM_CHANNEL_6

Compare channel 6 identifier

TIM_CHANNEL_ALL

Global Capture/compare channel identifier

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clear Input Source

TIM_CLEARINPUTSOURCE_NONE

OCREF_CLR is disabled

TIM_CLEARINPUTSOURCE_ETR

OCREF_CLR is connected to ETRF input

TIM_CLEARINPUTSOURCE_COMP1

OCREF_CLR_INT is connected to COMP1 output

TIM_CLEARINPUTSOURCE_COMP2

OCREF_CLR_INT is connected to COMP2 output

TIM_CLEARINPUTSOURCE_COMP3

OCREF_CLR_INT is connected to COMP3 output

TIM_CLEARINPUTSOURCE_COMP4

OCREF_CLR_INT is connected to COMP4 output

TIM_CLEARINPUTSOURCE_COMP5

OCREF_CLR_INT is connected to COMP5 output

TIM_CLEARINPUTSOURCE_COMP6

OCREF_CLR_INT is connected to COMP6 output

TIM_CLEARINPUTSOURCE_COMP7

OCREF_CLR_INT is connected to COMP7 output

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

Clock division: tDTS=tCK_INT

TIM_CLOCKDIVISION_DIV2

Clock division: tDTS=2*tCK_INT

TIM_CLOCKDIVISION_DIV4

Clock division: tDTS=4*tCK_INT

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2

External clock source mode 2

TIM_CLOCKSOURCE_INTERNAL

Internal clock source

TIM_CLOCKSOURCE_ITR0

External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1

External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2

External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3

External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_TI1ED

External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1

External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2

External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ETRMODE1

External clock source mode 1 (ETRF)

TIM_CLOCKSOURCE_ITR4

External clock source mode 1 (ITR4)

TIM_CLOCKSOURCE_ITR5

External clock source mode 1 (ITR5)

TIM_CLOCKSOURCE_ITR6

External clock source mode 1 (ITR6)

TIM_CLOCKSOURCE_ITR7

External clock source mode 1 (ITR7)

TIM_CLOCKSOURCE_ITR8

External clock source mode 1 (ITR8)

TIM_CLOCKSOURCE_ITR9

External clock source mode 1 (ITR9)

TIM_CLOCKSOURCE_ITR10

External clock source mode 1 (ITR10)

TIM_CLOCKSOURCE_ITR11

External clock source mode 1 (ITR11)

TIM Commutation Source

TIM_COMMUTATION_TRGI

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit or when an rising edge occurs on trigger input

TIM_COMMUTATION_SOFTWARE

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit

TIM Counter Mode

TIM_COUNTERMODE_UP

Counter used as up-counter

TIM_COUNTERMODE_DOWN

Counter used as down-counter

TIM_COUNTERMODE_CENTERALIGNED1

Center-aligned mode 1

TIM_COUNTERMODE_CENTERALIGNED2

Center-aligned mode 2

TIM_COUNTERMODE_CENTERALIGNED3

Center-aligned mode 3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_CCMR3

TIM_DMABASE_DTR2

TIM_DMABASE_ECR

TIM_DMABASE_TISEL

TIM_DMABASE_AF1

TIM_DMABASE_AF2

TIM_DMABASE_OR

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

The transfer is done to 1 register starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_2TRANSFERS

The transfer is done to 2 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_3TRANSFERS

The transfer is done to 3 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_4TRANSFERS

The transfer is done to 4 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_5TRANSFERS

The transfer is done to 5 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_6TRANSFERS

The transfer is done to 6 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_7TRANSFERS

The transfer is done to 7 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_8TRANSFERS

The transfer is done to 8 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_9TRANSFERS

The transfer is done to 9 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_10TRANSFERS

The transfer is done to 10 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_11TRANSFERS

The transfer is done to 11 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_12TRANSFERS

The transfer is done to 12 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_13TRANSFERS

The transfer is done to 13 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_14TRANSFERS

The transfer is done to 14 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_15TRANSFERS

The transfer is done to 15 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_16TRANSFERS

The transfer is done to 16 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_17TRANSFERS

The transfer is done to 17 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_18TRANSFERS

The transfer is done to 18 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_19TRANSFERS

The transfer is done to 19 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_20TRANSFERS

The transfer is done to 20 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_21TRANSFERS

The transfer is done to 21 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_22TRANSFERS

The transfer is done to 22 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_23TRANSFERS

The transfer is done to 23 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_24TRANSFERS

The transfer is done to 24 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_25TRANSFERS

The transfer is done to 25 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_26TRANSFERS

The transfer is done to 26 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM DMA Sources

TIM_DMA_UPDATE

DMA request is triggered by the update event

TIM_DMA_CC1

DMA request is triggered by the capture/compare macth 1 event

TIM_DMA_CC2

DMA request is triggered by the capture/compare macth 2 event event

TIM_DMA_CC3

DMA request is triggered by the capture/compare macth 3 event event

TIM_DMA_CC4

DMA request is triggered by the capture/compare macth 4 event event

TIM_DMA_COM

DMA request is triggered by the commutation event

TIM_DMA_TRIGGER

DMA request is triggered by the trigger event

TIM Encoder Input Polarity

TIM_ENCODERINPUTPOLARITY_RISING

Encoder input with rising edge polarity

TIM_ENCODERINPUTPOLARITY_FALLING

Encoder input with falling edge polarity

TIM Encoder Mode

TIM_ENCODERMODE_TI1

Quadrature encoder mode 1, x2 mode, counts up/down on TI1FP1 edge depending on TI2FP2 level

TIM_ENCODERMODE_TI2

Quadrature encoder mode 2, x2 mode, counts up/down on TI2FP2 edge depending on TI1FP1 level.

TIM_ENCODERMODE_TI12

Quadrature encoder mode 3, x4 mode, counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction, x2 mode

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED

Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED

Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1

No prescaler is used

TIM_ETRPRESCALER_DIV2

ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4

ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8

ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:

- Enable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_DISABLE_IT

Description:

- Disable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG

Description:

- Check whether the specified TIM interrupt flag is set or not.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_CLEAR_FLAG

Description:

- Clear the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE

Description:

- Check whether the specified TIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_UIFREMAP_ENABLE

Description:

- Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

Notes:

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

__HAL_TIM_UIFREMAP_DISABLE

Description:

- Disable update interrupt flag (UIF) remapping.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

__HAL_TIM_GET_UIFCPY

Description:

- Get update interrupt flag (UIF) copy status.

Parameters:

- __COUNTER__: Counter value.

Return value:

- The: state of UIFCPY (TRUE or FALSE). mode.

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:

- Set the TIM Prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:

- None

__HAL_TIM_SET_COUNTER

Description:

- Set the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_COUNTER

Description:

- Get the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:

- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:

- Get the TIM Input Capture prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: get input capture 1 prescaler value
  - TIM_CHANNEL_2: get input capture 2 prescaler value
  - TIM_CHANNEL_3: get input capture 3 prescaler value
  - TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- The: input capture prescaler can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_COMPARE

Description:

- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE

Description:

- Get the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value
  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD

Description:

- Set the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_DISABLE_OCxPRELOAD

Description:

- Reset the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_ENABLE_OCxFAST

Description:

- Enable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is enabled an active edge on the trigger input acts like a compare match on CCx output. Delay to sample the trigger input and to activate CCx output is reduced to 3 clock cycles. Fast mode acts only if the channel is configured in PWM1 or PWM2 mode.

__HAL_TIM_DISABLE_OCxFAST

Description:

- Disable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is disabled CCx output behaves normally depending on counter and CCRx values even when the trigger is ON. The minimum delay to activate CCx output when an active edge occurs on the trigger input is 5 clock cycles.

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

TIM Flag Definition

TIM_FLAG_UPDATE

Update interrupt flag

TIM_FLAG_CC1

Capture/Compare 1 interrupt flag

TIM_FLAG_CC2

Capture/Compare 2 interrupt flag

TIM_FLAG_CC3

Capture/Compare 3 interrupt flag

TIM_FLAG_CC4

Capture/Compare 4 interrupt flag

TIM_FLAG_CC5

Capture/Compare 5 interrupt flag

TIM_FLAG_CC6

Capture/Compare 6 interrupt flag

TIM_FLAG_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

Break interrupt flag

TIM_FLAG_BREAK2

Break 2 interrupt flag

TIM_FLAG_SYSTEM_BREAK

System Break interrupt flag

TIM_FLAG_CC1OF

Capture 1 overcapture flag

TIM_FLAG_CC2OF

Capture 2 overcapture flag

TIM_FLAG_CC3OF

Capture 3 overcapture flag

TIM_FLAG_CC4OF

Capture 4 overcapture flag

TIM_FLAG_IDX

Encoder index flag

TIM_FLAG_DIR

Direction change flag

TIM_FLAG_IERR

Index error flag

TIM_FLAG_TERR

Transition error flag

Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

TIM_GROUPCH5_OC1REFC

TIM_GROUPCH5_OC2REFC

TIM_GROUPCH5_OC3REFC

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

Capture triggered by rising edge on timer input

TIM_ICPOLARITY_FALLING

Capture triggered by falling edge on timer input

TIM_ICPOLARITY_BOTHEDGE

Capture triggered by both rising and falling edges on timer input

TIM Input Capture Prescaler

TIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2

Capture performed once every 2 events

TIM_ICPSC_DIV4

Capture performed once every 4 events

TIM_ICPSC_DIV8

Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity

TIM_INPUTCHANNELPOLARITY_RISING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

Update interrupt

TIM_IT_CC1

Capture/Compare 1 interrupt

TIM_IT_CC2

Capture/Compare 2 interrupt

TIM_IT_CC3

Capture/Compare 3 interrupt

TIM_IT_CC4

Capture/Compare 4 interrupt

TIM_IT_COM

Commutation interrupt

TIM_IT_TRIGGER

Trigger interrupt

TIM_IT_BREAK

Break interrupt

TIM_IT_IDX

Index interrupt

TIM_IT_DIR

Direction change interrupt

TIM_IT_IERR

Index error interrupt

TIM_IT_TERR

Transition error interrupt

TIM Lock level

TIM_LOCKLEVEL_OFF

LOCK OFF

TIM_LOCKLEVEL_1

LOCK Level 1

TIM_LOCKLEVEL_2

LOCK Level 2

TIM_LOCKLEVEL_3

LOCK Level 3

TIM Master Mode Selection

TIM_TRGO_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO)

TIM_TRGO_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO)

TIM_TRGO_UPDATE

Update event is used as trigger output (TRGO)

TIM_TRGO_OC1

Capture or a compare match 1 is used as trigger output (TRGO)

TIM_TRGO_OC1REF

OC1REF signal is used as trigger output (TRGO)

TIM_TRGO_OC2REF

OC2REF signal is used as trigger output(TRGO)

TIM_TRGO_OC3REF

OC3REF signal is used as trigger output(TRGO)

TIM_TRGO_OC4REF

OC4REF signal is used as trigger output(TRGO)

TIM_TRGO_ENCODER_CLK

Encoder clock is used as trigger output(TRGO)

TIM Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO2)

TIM_TRGO2_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO2)

TIM_TRGO2_UPDATE

Update event is used as trigger output (TRGO2)

TIM_TRGO2_OC1

Capture or a compare match 1 is used as trigger output (TRGO2)

TIM_TRGO2_OC1REF

OC1REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC2REF

OC2REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC3REF

OC3REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF

OC4REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC5REF

OC5REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC6REF

OC6REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF_RISINGFALLING

OC4REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC6REF_RISINGFALLING

OC6REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

OC4REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

OC4REF rising or OC6REF falling edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

No action

TIM_MASTERSLAVEMODE_DISABLE

Master/slave mode is selected

TIM One Pulse Mode

TIM_OPMODE_SINGLE

Counter stops counting at the next update event

TIM_OPMODE_REPETITIVE

Counter is not stopped at update event

TIM OSSI OffState Selection for Idle mode state

TIM_OSSI_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSI_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM OSSR OffState Selection for Run mode state

TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSR_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM Output Compare and PWM Modes

TIM_OCMODE_TIMING

Frozen

TIM_OCMODE_ACTIVE

Set channel to active level on match

TIM_OCMODE_INACTIVE

Set channel to inactive level on match

TIM_OCMODE_TOGGLE

Toggle

TIM_OCMODE_PWM1

PWM mode 1

TIM_OCMODE_PWM2

PWM mode 2

TIM_OCMODE_FORCED_ACTIVE

Force active level

TIM_OCMODE_FORCED_INACTIVE

Force inactive level

TIM_OCMODE_RETRIGERRABLE_OPM1

Retrigerrable OPM mode 1

TIM_OCMODE_RETRIGERRABLE_OPM2

Retrigerrable OPM mode 2

TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

TIM_OCMODE_ASSYMETRIC_PWM1

Asymmetric PWM mode 1

TIM_OCMODE_ASSYMETRIC_PWM2

Asymmetric PWM mode 2

TIM_OCMODE_PULSE_ON_COMPARE

Pulse on compare (CH3&CH4 only)

TIM_OCMODE_DIRECTION_OUTPUT

Direction output (CH3&CH4 only)

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

Output Idle state: OCx=1 when MOE=0

TIM_OCIDLESTATE_RESET

Output Idle state: OCx=0 when MOE=0

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

Complementary output Idle state: OCxN=1 when MOE=0

TIM_OCNIDLESTATE_RESET

Complementary output Idle state: OCxN=0 when MOE=0

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

Capture/Compare complementary output polarity

TIM_OCNPOLARITY_LOW

Capture/Compare complementary output polarity

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

OCxN is disabled

TIM_OUTPUTNSTATE_ENABLE

OCxN is enabled

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

Capture/Compare output polarity

TIM_OCPOLARITY_LOW

Capture/Compare output polarity

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

Capture/Compare 1 output disabled

TIM_OUTPUTSTATE_ENABLE

Capture/Compare 1 output enabled

TIM Output Fast State

TIM_OCFAST_DISABLE

Output Compare fast disable

TIM_OCFAST_ENABLE

Output Compare fast enable

TIM Slave mode

TIM_SLAVEMODE_DISABLE

Slave mode disabled

TIM_SLAVEMODE_RESET

Reset Mode

TIM_SLAVEMODE_GATED

Gated Mode

TIM_SLAVEMODE_TRIGGER

Trigger Mode

TIM_SLAVEMODE_EXTERNAL1

External Clock Mode 1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode

TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

The TIMx_CH1 pin is connected to TI1 input

TIM_TI1SELECTION_XORCOMBINATION

The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

Internal Trigger 0 (ITR0)

TIM_TS_ITR1

Internal Trigger 1 (ITR1)

TIM_TS_ITR2

Internal Trigger 2 (ITR2)

TIM_TS_ITR3

Internal Trigger 3 (ITR3)

TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED)

TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1)

TIM_TS_TI2FP2

Filtered Timer Input 2 (TI2FP2)

TIM_TS_ETRF

Filtered External Trigger input (ETRF)

TIM_TS_ITR4

Internal Trigger 4 (ITR9)

TIM_TS_ITR5

Internal Trigger 5 (ITR5)

TIM_TS_ITR6

Internal Trigger 6 (ITR6)

TIM_TS_ITR7

Internal Trigger 7 (ITR7)

TIM_TS_ITR8

Internal Trigger 8 (ITR8)

TIM_TS_ITR9

Internal Trigger 9 (ITR9)

TIM_TS_ITR10

Internal Trigger 10 (ITR10)

TIM_TS_ITR11

Internal Trigger 11 (ITR11)

TIM_TS_NONE

No trigger selected

TIM Update Interrupt Flag Remap

TIM_UIFREMAP_DISABLE

Update interrupt flag remap disabled

TIM_UIFREMAP_ENABLE

Update interrupt flag remap enabled

59 HAL TIM Extension Driver

59.1 TIMEx Firmware driver registers structures

59.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32g4xx_hal_tim_ex.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t Commutation_Delay***

Field Documentation

- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

59.1.2 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32g4xx_hal_tim_ex.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Enable***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t TIMEx_BreakInputConfigTypeDef::Source***
  Specifies the source of the timer break input. This parameter can be a value of ***TIMEx_Break_Input_Source***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Enable***
  Specifies whether or not the break input source is enabled. This parameter can be a value of ***TIMEx_Break_Input_Source_Enable***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Polarity***
  Specifies the break input source polarity. This parameter can be a value of ***TIMEx_Break_Input_Source_Polarity***

59.1.3 TIMEx_EncoderIndexConfigTypeDef

TIMEx_EncoderIndexConfigTypeDef is defined in the stm32g4xx_hal_tim_ex.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t Prescaler***
- ***uint32_t Filter***
- ***FunctionalState FirstIndexEnable***
- ***uint32_t Position***

- ***uint32_t Direction***

Field Documentation

- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Polarity***
  TIM Encoder index polarity. This parameter can be a value of ***TIMEx_Encoder_Index_Polarity***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Prescaler***
  TIM Encoder index prescaler. This parameter can be a value of ***TIMEx_Encoder_Index_Prescaler***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Filter***
  TIM Encoder index filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***FunctionalState TIMEx_EncoderIndexConfigTypeDef::FirstIndexEnable***
  Specifies whether or not the encoder first index is enabled. This parameter value can be ENABLE or DISABLE.
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Position***
  Specifies in which AB input configuration the index event resets the counter. This parameter can be a value of ***TIMEx_Encoder_Index_Position***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Direction***
  Specifies in which counter direction the index event resets the counter. This parameter can be a value of ***TIMEx_Encoder_Index_Direction***

59.2 TIMEx Firmware driver API description

The following section lists the various functions of the TIMEx library.

59.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
5. In case of Pulse on compare, configure pulse length and delay
6. Encoder index configuration

59.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Hall Sensor output : HAL_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
   – HAL_TIMEx_HallSensor_Init() and HAL_TIMEx_ConfigCommutEvent(): to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).

5. In case of Pulse On Compare:
   – HAL_TIMEx_OC_ConfigPulseOnCompare(): to configure pulse width and prescaler
6. Activate the TIM peripheral using one of the start functions:
   – Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   – Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   – Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   – Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

59.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:

- ***HAL_TIMEx_HallSensor_Init***
- ***HAL_TIMEx_HallSensor_DeInit***
- ***HAL_TIMEx_HallSensor_MspInit***
- ***HAL_TIMEx_HallSensor_MspDeInit***
- ***HAL_TIMEx_HallSensor_Start***
- ***HAL_TIMEx_HallSensor_Stop***
- ***HAL_TIMEx_HallSensor_Start_IT***
- ***HAL_TIMEx_HallSensor_Stop_IT***
- ***HAL_TIMEx_HallSensor_Start_DMA***
- ***HAL_TIMEx_HallSensor_Stop_DMA***

59.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:

- ***HAL_TIMEx_OCN_Start***
- ***HAL_TIMEx_OCN_Stop***
- ***HAL_TIMEx_OCN_Start_IT***
- ***HAL_TIMEx_OCN_Stop_IT***
- ***HAL_TIMEx_OCN_Start_DMA***
- ***HAL_TIMEx_OCN_Stop_DMA***

59.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- ***HAL_TIMEx_PWMN_Start***
- ***HAL_TIMEx_PWMN_Stop***
- ***HAL_TIMEx_PWMN_Start_IT***
- ***HAL_TIMEx_PWMN_Stop_IT***
- ***HAL_TIMEx_PWMN_Start_DMA***
- ***HAL_TIMEx_PWMN_Stop_DMA***

59.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- ***HAL_TIMEx_OnePulseN_Start***
- ***HAL_TIMEx_OnePulseN_Stop***
- ***HAL_TIMEx_OnePulseN_Start_IT***
- ***HAL_TIMEx_OnePulseN_Stop_IT***

59.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Select timer input source.
- Enable or disable channel grouping.
- Configure Pulse on compare.
- Configure Encoder index.

This section contains the following APIs:

- *HAL_TIMEx_ConfigCommutEvent*
- *HAL_TIMEx_ConfigCommutEvent_IT*
- *HAL_TIMEx_ConfigCommutEvent_DMA*
- *HAL_TIMEx_MasterConfigSynchronization*
- *HAL_TIMEx_ConfigBreakDeadTime*
- *HAL_TIMEx_ConfigBreakInput*
- *HAL_TIMEx_RemapConfig*
- *HAL_TIMEx_TISelection*
- *HAL_TIMEx_GroupChannel5*
- *HAL_TIMEx_DisarmBreakInput*
- *HAL_TIMEx_ReArmBreakInput*
- *HAL_TIMEx_DitheringEnable*
- *HAL_TIMEx_DitheringDisable*
- *HAL_TIMEx_OC_ConfigPulseOnCompare*
- *HAL_TIMEx_ConfigSlaveModePreload*
- *HAL_TIMEx_EnableSlaveModePreload*
- *HAL_TIMEx_DisableSlaveModePreload*
- *HAL_TIMEx_EnableDeadTimePreload*
- *HAL_TIMEx_DisableDeadTimePreload*
- *HAL_TIMEx_ConfigDeadTime*
- *HAL_TIMEx_ConfigAsymmetricalDeadTime*
- *HAL_TIMEx_EnableAsymmetricalDeadTime*
- *HAL_TIMEx_DisableAsymmetricalDeadTime*
- *HAL_TIMEx_ConfigEncoderIndex*
- *HAL_TIMEx_EnableEncoderIndex*
- *HAL_TIMEx_DisableEncoderIndex*
- *HAL_TIMEx_EnableEncoderFirstIndex*
- *HAL_TIMEx_DisableEncoderFirstIndex*

59.2.8 Extended Callbacks functions

This section provides Extended TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- *HAL_TIMEx_CommutCallback*
- *HAL_TIMEx_CommutHalfCpltCallback*
- *HAL_TIMEx_BreakCallback*
- *HAL_TIMEx_Break2Callback*
- *HAL_TIMEx_EncoderIndexCallback*
- *HAL_TIMEx_DirectionChangeCallback*
- *HAL_TIMEx_IndexErrorCallback*
- *HAL_TIMEx_TransitionErrorCallback*

59.2.9 Extended Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_TIMEx_HallSensor_GetState*
- *HAL_TIMEx_GetChannelNState*

59.2.10 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)

Function description

Initializes the TIM Hall Sensor Interface and initialize the associated handle.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **sConfig:** TIM Hall Sensor configuration structure

Return values

- **HAL:** status

Notes

- When the timer instance is initialized in Hall Sensor Interface mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIMEx_HallSensor_DeInit

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Hall Sensor interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_MspInit

Function name

void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_MspDeInit

Function name

void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM OC handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_ConfigCommutEvent

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected (*)
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR9: Internal trigger 9 selected (*)
  - TIM_TS_ITR10: Internal trigger 10 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with interrupt.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected (*)
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR9: Internal trigger 9 selected (*)
  - TIM_TS_ITR10: Internal trigger 10 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with DMA.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected (*)
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR9: Internal trigger 9 selected (*)
  - TIM_TS_ITR10: Internal trigger 10 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_MasterConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)

Function description

Configures the TIM in master mode.

Parameters

- **htim:** TIM handle.
- **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description

Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

Parameters

- **htim:** TIM handle
- **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- **HAL:** status

Notes

- Interrupts can be generated when an active level is detected on the break input, the break 2 input or the system break input. Break interrupt can be enabled by calling the __HAL_TIM_ENABLE_IT macro.

HAL_TIMEx_ConfigBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput, TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)

Function description

Configures the break input source.

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to configure This parameter can be one of the following values:
  - TIM_BREAKINPUT_BRK: Timer break input
  - TIM_BREAKINPUT_BRK2: Timer break 2 input
- **sBreakInputConfig:** Break input source configuration

Return values

- **HAL:** status

HAL_TIMEx_GroupChannel5

Function name

HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)

Function description

Group channel 5 and channel 1, 2 or 3.

Parameters

- **htim:** TIM handle.
- **Channels:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values

- **HAL:** status

HAL_TIMEx_RemapConfig

Function name

HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)

Function description

Configures the TIMx Remapping input capabilities.

Parameters

- **htim:** TIM handle.

- **Remap:** specifies the TIM remapping source. For TIM1, the parameter can take one of the following values:
  - TIM_TIM1_ETR_GPIO TIM1 ETR is connected to GPIO
  - TIM_TIM1_ETR_COMP1 TIM1 ETR is connected to COMP1 output
  - TIM_TIM1_ETR_COMP2 TIM1 ETR is connected to COMP2 output
  - TIM_TIM1_ETR_COMP3 TIM1 ETR is connected to COMP3 output
  - TIM_TIM1_ETR_COMP4 TIM1 ETR is connected to COMP4 output
  - TIM_TIM1_ETR_COMP5 TIM1 ETR is connected to COMP5 output (*)
  - TIM_TIM1_ETR_COMP6 TIM1 ETR is connected to COMP6 output (*)
  - TIM_TIM1_ETR_COMP7 TIM1 ETR is connected to COMP7 output (*)
  - TIM_TIM1_ETR_ADC1_AWD1 TIM1 ETR is connected to ADC1 AWD1
  - TIM_TIM1_ETR_ADC1_AWD2 TIM1 ETR is connected to ADC1 AWD2
  - TIM_TIM1_ETR_ADC1_AWD3 TIM1 ETR is connected to ADC1 AWD3
  - TIM_TIM1_ETR_ADC4_AWD1 TIM1 ETR is connected to ADC4 AWD1 (*)
  - TIM_TIM1_ETR_ADC4_AWD2 TIM1 ETR is connected to ADC4 AWD2 (*)
  - TIM_TIM1_ETR_ADC4_AWD3 TIM1 ETR is connected to ADC4 AWD3 (*)

  For TIM2, the parameter can take one of the following values:
  - TIM_TIM2_ETR_GPIO TIM2 ETR is connected to GPIO
  - TIM_TIM2_ETR_COMP1 TIM2 ETR is connected to COMP1 output
  - TIM_TIM2_ETR_COMP2 TIM2 ETR is connected to COMP2 output
  - TIM_TIM2_ETR_COMP3 TIM2 ETR is connected to COMP3 output
  - TIM_TIM2_ETR_COMP4 TIM2 ETR is connected to COMP4 output
  - TIM_TIM2_ETR_COMP5 TIM2 ETR is connected to COMP5 output (*)
  - TIM_TIM2_ETR_COMP6 TIM2 ETR is connected to COMP6 output (*)
  - TIM_TIM2_ETR_COMP7 TIM2 ETR is connected to COMP7 output (*)
  - TIM_TIM2_ETR_TIM3_ETR TIM2 ETR is connected to TIM3 ETR pin
  - TIM_TIM2_ETR_TIM4_ETR TIM2 ETR is connected to TIM4 ETR pin
  - TIM_TIM2_ETR_TIM5_ETR TIM2 ETR is connected to TIM5 ETR pin (*)
  - TIM_TIM2_ETR_LSE

  For TIM3, the parameter can take one of the following values:
  - TIM_TIM3_ETR_GPIO TIM3 ETR is connected to GPIO
  - TIM_TIM3_ETR_COMP1 TIM3 ETR is connected to COMP1 output
  - TIM_TIM3_ETR_COMP2 TIM3 ETR is connected to COMP2 output
  - TIM_TIM3_ETR_COMP3 TIM3 ETR is connected to COMP3 output
  - TIM_TIM3_ETR_COMP4 TIM3 ETR is connected to COMP4 output
  - TIM_TIM3_ETR_COMP5 TIM3 ETR is connected to COMP5 output (*)
  - TIM_TIM3_ETR_COMP6 TIM3 ETR is connected to COMP6 output (*)
  - TIM_TIM3_ETR_COMP7 TIM3 ETR is connected to COMP7 output (*)
  - TIM_TIM3_ETR_TIM2_ETR TIM3 ETR is connected to TIM2 ETR pin
  - TIM_TIM3_ETR_TIM4_ETR TIM3 ETR is connected to TIM4 ETR pin
  - TIM_TIM3_ETR_ADC2_AWD1 TIM3 ETR is connected to ADC2 AWD1
  - TIM_TIM3_ETR_ADC2_AWD2 TIM3 ETR is connected to ADC2 AWD2
  - TIM_TIM3_ETR_ADC2_AWD3 TIM3 ETR is connected to ADC2 AWD3

  For TIM4, the parameter can take one of the following values:
  - TIM_TIM4_ETR_GPIO TIM4 ETR is connected to GPIO
  - TIM_TIM4_ETR_COMP1 TIM4 ETR is connected to COMP1 output
  - TIM_TIM4_ETR_COMP2 TIM4 ETR is connected to COMP2 output
  - TIM_TIM4_ETR_COMP3 TIM4 ETR is connected to COMP3 output
  - TIM_TIM4_ETR_COMP4 TIM4 ETR is connected to COMP4 output
  - TIM_TIM4_ETR_COMP5 TIM4 ETR is connected to COMP5 output (*)
  - TIM_TIM4_ETR_COMP6 TIM4 ETR is connected to COMP6 output (*)
  - TIM_TIM4_ETR_COMP7 TIM4 ETR is connected to COMP7 output (*)

- (**) Register not available in all devices.

Return values

- **HAL:** status

HAL_TIMEx_TISelection

Function name

HAL_StatusTypeDef HAL_TIMEx_TISelection (TIM_HandleTypeDef * htim, uint32_t TISelection, uint32_t Channel)

Function description

Select the timer input source.

Parameters

- **htim:** TIM handle.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TI1 input channel
  - TIM_CHANNEL_2: TI2 input channel
  - TIM_CHANNEL_3: TI3 input channel
  - TIM_CHANNEL_4: TI4 input channel

- **TISelection:** specifies the timer input source For TIM1 this parameter can be one of the following values:
  - TIM_TIM1_TI1_GPIO: TIM1 TI1 is connected to GPIO
  - TIM_TIM1_TI1_COMP1: TIM1 TI1 is connected to COMP1 output
  - TIM_TIM1_TI1_COMP2: TIM1 TI1 is connected to COMP2 output
  - TIM_TIM1_TI1_COMP3: TIM1 TI1 is connected to COMP3 output
  - TIM_TIM1_TI1_COMP4: TIM1 TI1 is connected to COMP4 output

  For TIM2 this parameter can be one of the following values:
  - TIM_TIM2_TI1_GPIO: TIM2 TI1 is connected to GPIO
  - TIM_TIM2_TI1_COMP1: TIM2 TI1 is connected to COMP1 output
  - TIM_TIM2_TI1_COMP2: TIM2 TI1 is connected to COMP2 output
  - TIM_TIM2_TI1_COMP3: TIM2 TI1 is connected to COMP3 output
  - TIM_TIM2_TI1_COMP4: TIM2 TI1 is connected to COMP4 output
  - TIM_TIM2_TI1_COMP5: TIM2 TI1 is connected to COMP5 output (*)
  - TIM_TIM2_TI2_GPIO: TIM1 TI2 is connected to GPIO
  - TIM_TIM2_TI2_COMP1: TIM2 TI2 is connected to COMP1 output
  - TIM_TIM2_TI2_COMP2: TIM2 TI2 is connected to COMP2 output
  - TIM_TIM2_TI2_COMP3: TIM2 TI2 is connected to COMP3 output
  - TIM_TIM2_TI2_COMP4: TIM2 TI2 is connected to COMP4 output
  - TIM_TIM2_TI2_COMP6: TIM2 TI2 is connected to COMP6 output (*)
  - TIM_TIM2_TI3_GPIO: TIM2 TI3 is connected to GPIO
  - TIM_TIM2_TI3_COMP4: TIM2 TI3 is connected to COMP4 output
  - TIM_TIM2_TI4_GPIO: TIM2 TI4 is connected to GPIO
  - TIM_TIM2_TI4_COMP1: TIM2 TI4 is connected to COMP1 output
  - TIM_TIM2_TI4_COMP2: TIM2 TI4 is connected to COMP2 output

  For TIM3 this parameter can be one of the following values:
  - TIM_TIM3_TI1_GPIO: TIM3 TI1 is connected to GPIO
  - TIM_TIM3_TI1_COMP1: TIM3 TI1 is connected to COMP1 output
  - TIM_TIM3_TI1_COMP2: TIM3 TI1 is connected to COMP2 output
  - TIM_TIM3_TI1_COMP3: TIM3 TI1 is connected to COMP3 output
  - TIM_TIM3_TI1_COMP4: TIM3 TI1 is connected to COMP4 output
  - TIM_TIM3_TI1_COMP5: TIM3 TI1 is connected to COMP5 output (*)
  - TIM_TIM3_TI1_COMP6: TIM3 TI1 is connected to COMP6 output (*)
  - TIM_TIM3_TI1_COMP7: TIM3 TI1 is connected to COMP7 output (*)
  - TIM_TIM3_TI2_GPIO: TIM3 TI2 is connected to GPIO
  - TIM_TIM3_TI2_COMP1: TIM3 TI2 is connected to COMP1 output
  - TIM_TIM3_TI2_COMP2: TIM3 TI2 is connected to COMP2 output
  - TIM_TIM3_TI2_COMP3: TIM3 TI2 is connected to COMP3 output
  - TIM_TIM3_TI2_COMP4: TIM3 TI2 is connected to COMP4 output
  - TIM_TIM3_TI2_COMP5: TIM3 TI2 is connected to COMP5 output (*)
  - TIM_TIM3_TI2_COMP6: TIM3 TI2 is connected to COMP6 output (*)
  - TIM_TIM3_TI2_COMP7: TIM3 TI2 is connected to COMP7 output (*)
  - TIM_TIM3_TI3_GPIO: TIM3 TI3 is connected to GPIO
  - TIM_TIM3_TI3_COMP3: TIM3 TI3 is connected to COMP3 output

  For TIM4 this parameter can be one of the following values:
  - TIM_TIM4_TI1_GPIO: TIM4 TI1 is connected to GPIO
  - TIM_TIM4_TI1_COMP1: TIM4 TI1 is connected to COMP1 output
  - TIM_TIM4_TI1_COMP2: TIM4 TI1 is connected to COMP2 output
  - TIM_TIM4_TI1_COMP3: TIM4 TI1 is connected to COMP3 output
  - TIM_TIM4_TI1_COMP4: TIM4 TI1 is connected to COMP4 output
  - TIM_TIM4_TI1_COMP5: TIM4 TI1 is connected to COMP5 output (*)

- (**) Register not available in all devices.

Return values

- **HAL:** status

HAL_TIMEx_DisarmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_DisarmBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

Disarm the designated break input (when it operates in bidirectional mode).

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to disarm This parameter can be one of the following values:
  - TIM_BREAKINPUT_BRK: Timer break input
  - TIM_BREAKINPUT_BRK2: Timer break 2 input

Return values

- **HAL:** status

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

HAL_TIMEx_ReArmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ReArmBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

Arm the designated break input (when it operates in bidirectional mode).

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to arm This parameter can be one of the following values:
  - TIM_BREAKINPUT_BRK: Timer break input
  - TIM_BREAKINPUT_BRK2: Timer break 2 input

Return values

- **HAL:** status

Notes

- Arming is possible at anytime, even if fault is present.
- Break input is automatically armed as soon as MOE bit is set.

HAL_TIMEx_DitheringEnable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringEnable (TIM_HandleTypeDef * htim)

Function description

Enable dithering.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

Notes

- Main usage is PWM mode
- This function must be called when timer is stopped or disabled (CEN =0)
- If dithering is activated, pay attention to ARR, CCRx, CNT interpretation: CNT: only CNT[11:0] holds the non-dithered part for 16b timers (or CNT[26:0] for 32b timers)ARR: ARR[15:4] holds the non-dithered part, and ARR[3:0] the dither part for 16b timersCCRx: CCRx[15:4] holds the non-dithered part, and CCRx[3:0] the dither part for 16b timersARR and CCRx values are limited to 0xFFEF in dithering mode for 16b timers (corresponds to 4094 for the integer part and 15 for the dithered part).
- Macros __HAL_TIM_CALC_PERIOD_DITHER() __HAL_TIM_CALC_DELAY_DITHER() __HAL_TIM_CALC_PULSE_DITHER() can be used to calculate period (ARR) and delay (CCRx) value.
- Enabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part. So it may be necessary to read ARR value or CCRx value with macros __HAL_TIM_GET_AUTORELOAD() __HAL_TIM_GET_COMPARE() and if necessary update Init structure field htim->Init.Period .

HAL_TIMEx_DitheringDisable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringDisable (TIM_HandleTypeDef * htim)

Function description

Disable dithering.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

Notes

- This function must be called when timer is stopped or disabled (CEN =0)
- If dithering is activated, pay attention to ARR, CCRx, CNT interpretation: CNT: only CNT[11:0] holds the non-dithered part for 16b timers (or CNT[26:0] for 32b timers)ARR: ARR[15:4] holds the non-dithered part, and ARR[3:0] the dither part for 16b timersCCRx: CCRx[15:4] holds the non-dithered part, and CCRx[3:0] the dither part for 16b timersARR and CCRx values are limited to 0xFFEF in dithering mode (corresponds to 4094 for the integer part and 15 for the dithered part).
- Disabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part. So it may be necessary to read ARR value or CCRx value with macros __HAL_TIM_GET_AUTORELOAD() __HAL_TIM_GET_COMPARE() and if necessary update Init structure field htim->Init.Period .

HAL_TIMEx_OC_ConfigPulseOnCompare

Function name

HAL_StatusTypeDef HAL_TIMEx_OC_ConfigPulseOnCompare (TIM_HandleTypeDef * htim, uint32_t PulseWidthPrescaler, uint32_t PulseWidth)

Function description

Initializes the pulse on compare pulse width and pulse prescaler.

Parameters

- **htim:** TIM Output Compare handle
- **PulseWidthPrescaler:** Pulse width prescaler This parameter can be a number between Min_Data = 0x0 and Max_Data = 0x7
- **PulseWidth:** Pulse width This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

Return values

- **HAL:** status

HAL_TIMEx_ConfigSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigSlaveModePreload (TIM_HandleTypeDef * htim, uint32_t Source)

Function description

Configure preload source of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle
- **Source:** Source of slave mode selection preload This parameter can be one of the following values:
  - TIM_SMS_PRELOAD_SOURCE_UPDATE: Timer update event is used as source of Slave Mode Selection preload
  - TIM_SMS_PRELOAD_SOURCE_INDEX: Timer index event is used as source of Slave Mode Selection preload

Return values

- **HAL:** status

HAL_TIMEx_EnableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

Enable preload of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

Disable preload of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_EnableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

Enable deadtime preload.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

Disable deadtime preload.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigDeadTime (TIM_HandleTypeDef * htim, uint32_t Deadtime)

Function description

Configure deadtime.

Parameters

- **htim:** TIM handle
- **Deadtime:** Deadtime value

Return values

- **HAL:** status

Notes

- This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

HAL_TIMEx_ConfigAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigAsymmetricalDeadTime (TIM_HandleTypeDef * htim, uint32_t FallingDeadtime)

Function description

Configure asymmetrical deadtime.

Parameters

- **htim:** TIM handle
- **FallingDeadtime:** Falling edge deadtime value

Return values

- **HAL:** status

Notes

- This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

HAL_TIMEx_EnableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

Enable asymmetrical deadtime.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

Disable asymmetrical deadtime.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_ConfigEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigEncoderIndex (TIM_HandleTypeDef * htim, TIMEx_EncoderIndexConfigTypeDef * sEncoderIndexConfig)

Function description

Configures the encoder index.

Parameters

- **htim:** TIM handle.
- **sEncoderIndexConfig:** Encoder index configuration

Return values

- **HAL:** status

Notes

- warning in case of encoder mode clock plus direction TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1 or TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2 Direction must be set to TIM_ENCODERINDEX_DIRECTION_UP_DOWN

HAL_TIMEx_EnableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

Enable encoder index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

Disable encoder index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_EnableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

Enable encoder first index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

Disable encoder first index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_CommutCallback

Function name

void HAL_TIMEx_CommutCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_CommutHalfCpltCallback

Function name

void HAL_TIMEx_CommutHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_BreakCallback

Function name

void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)

Function description

Hall Break detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_Break2Callback

Function name

void HAL_TIMEx_Break2Callback (TIM_HandleTypeDef * htim)

Function description

Hall Break2 detection callback in non blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_EncoderIndexCallback

Function name

void HAL_TIMEx_EncoderIndexCallback (TIM_HandleTypeDef * htim)

Function description

Encoder index callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_DirectionChangeCallback

Function name

void HAL_TIMEx_DirectionChangeCallback (TIM_HandleTypeDef * htim)

Function description

Direction change callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_IndexErrorCallback

Function name

void HAL_TIMEx_IndexErrorCallback (TIM_HandleTypeDef * htim)

Function description

Index error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_TransitionErrorCallback

Function name

void HAL_TIMEx_TransitionErrorCallback (TIM_HandleTypeDef * htim)

Function description

Transition error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Hall Sensor interface handle state.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** state

HAL_TIMEx_GetChannelNState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIMEx_GetChannelNState (TIM_HandleTypeDef * htim, uint32_t ChannelN)

Function description

Return actual state of the TIM complementary channel.

Parameters

- **htim:** TIM handle
- **ChannelN:** TIM Complementary channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4

Return values

- **TIM:** Complementary channel state

TIMEx_DMACommutationCplt

Function name

void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIMEx_DMACommutationHalfCplt

Function name

void TIMEx_DMACommutationHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

59.3 TIMEx Firmware driver defines

The following section lists the various define and macros of the module.

59.3.1 TIMEx

TIMEx

TIM Extended Break input

TIM_BREAKINPUT_BRK

TIM_BREAKINPUT_BRK2

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

TIM_BREAKINPUTSOURCE_COMP1

TIM_BREAKINPUTSOURCE_COMP2

TIM_BREAKINPUTSOURCE_COMP3

TIM_BREAKINPUTSOURCE_COMP4

TIM_BREAKINPUTSOURCE_COMP5

TIM_BREAKINPUTSOURCE_COMP6

TIM_BREAKINPUTSOURCE_COMP7

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

TIM_BREAKINPUTSOURCE_ENABLE

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

TIM Extended Encoder index direction

TIM_ENCODERINDEX_DIRECTION_UP_DOWN

Index resets the counter whatever the direction

TIM_ENCODERINDEX_DIRECTION_UP

Index resets the counter when up-counting only

TIM_ENCODERINDEX_DIRECTION_DOWN

Index resets the counter when down-counting only

TIM Extended Encoder index polarity

TIM_ENCODERINDEX_POLARITY_INVERTED

Polarity for ETRx pin

TIM_ENCODERINDEX_POLARITY_NONINVERTED

Polarity for ETRx pin

TIM Extended Encoder index position

TIM_ENCODERINDEX_POSITION_00

Encoder index position is AB=00

TIM_ENCODERINDEX_POSITION_01

Encoder index position is AB=01

TIM_ENCODERINDEX_POSITION_10

Encoder index position is AB=10

TIM_ENCODERINDEX_POSITION_11

Encoder index position is AB=11

TIM_ENCODERINDEX_POSITION_0

In directional clock mode or clock plus direction mode, index resets the counter when clock is 0

TIM_ENCODERINDEX_POSITION_1

In directional clock mode or clock plus direction mode, index resets the counter when clock is 1

TIM Extended Encodder index prescaler

TIM_ENCODERINDEX_PRESCALER_DIV1

No prescaler is used

TIM_ENCODERINDEX_PRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_ENCODERINDEX_PRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_ENCODERINDEX_PRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Extended Exported Macros

__HAL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PSC(80000000, 1000000);

__HAL_TIM_CALC_PERIOD

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD(1000000, 0, 10000);

__HAL_TIM_CALC_PERIOD_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER(1000000, 0, 10000); This macro should be used only if ditehring is already enabled

__HAL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PULSE(1000000, 0, 10);

__HAL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PULSE_DITHER(1000000, 0, 10); This macro should be used only if ditehring is already enabled

__HAL_TIM_CALC_PERIOD_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_BY_DELAY(1000000, 0, 10, 20);

__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY(1000000, 0, 10, 20); This macro should be used only if ditehring is already enabled

TIM Extended Remapping

TIM_TIM1_ETR_GPIO

TIM_TIM1_ETR_COMP1

TIM_TIM1_ETR_COMP2

TIM_TIM1_ETR_COMP3

TIM_TIM1_ETR_COMP4

TIM_TIM1_ETR_COMP5

TIM_TIM1_ETR_COMP6

TIM_TIM1_ETR_COMP7

TIM_TIM1_ETR_ADC1_AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM_TIM1_ETR_ADC4_AWD1

TIM_TIM1_ETR_ADC4_AWD2

TIM_TIM1_ETR_ADC4_AWD3

TIM_TIM2_ETR_GPIO

TIM_TIM2_ETR_COMP1

TIM_TIM2_ETR_COMP2

TIM_TIM2_ETR_COMP3

TIM_TIM2_ETR_COMP4

TIM_TIM2_ETR_COMP5

TIM_TIM2_ETR_COMP6

TIM_TIM2_ETR_COMP7

TIM_TIM2_ETR_TIM3_ETR

TIM_TIM2_ETR_TIM4_ETR

TIM_TIM2_ETR_TIM5_ETR

TIM_TIM2_ETR_LSE

TIM_TIM3_ETR_GPIO

TIM_TIM3_ETR_COMP1

TIM_TIM3_ETR_COMP2

TIM_TIM3_ETR_COMP3

TIM_TIM3_ETR_COMP4

TIM_TIM3_ETR_COMP5

TIM_TIM3_ETR_COMP6

TIM_TIM3_ETR_COMP7

TIM_TIM3_ETR_TIM2_ETR

TIM_TIM3_ETR_TIM4_ETR

TIM_TIM3_ETR_ADC2_AWD1

TIM_TIM3_ETR_ADC2_AWD2

TIM_TIM3_ETR_ADC2_AWD3

TIM_TIM4_ETR_GPIO

TIM_TIM4_ETR_COMP1

TIM_TIM4_ETR_COMP2

TIM_TIM4_ETR_COMP3

TIM_TIM4_ETR_COMP4

TIM_TIM4_ETR_COMP5

TIM_TIM4_ETR_COMP6

TIM_TIM4_ETR_COMP7

TIM_TIM4_ETR_TIM3_ETR

TIM_TIM4_ETR_TIM5_ETR

TIM_TIM5_ETR_GPIO

TIM_TIM5_ETR_COMP1

TIM_TIM5_ETR_COMP2

TIM_TIM5_ETR_COMP3

TIM_TIM5_ETR_COMP4

TIM_TIM5_ETR_COMP5

TIM_TIM5_ETR_COMP6

TIM_TIM5_ETR_COMP7

TIM_TIM5_ETR_TIM2_ETR

TIM_TIM5_ETR_TIM3_ETR

TIM_TIM8_ETR_GPIO

TIM_TIM8_ETR_COMP1

TIM_TIM8_ETR_COMP2

TIM_TIM8_ETR_COMP3

TIM_TIM8_ETR_COMP4

TIM_TIM8_ETR_COMP5

TIM_TIM8_ETR_COMP6

TIM_TIM8_ETR_COMP7

TIM_TIM8_ETR_ADC2_AWD1

TIM_TIM8_ETR_ADC2_AWD2

TIM_TIM8_ETR_ADC2_AWD3

TIM_TIM8_ETR_ADC3_AWD1

TIM_TIM8_ETR_ADC3_AWD2

TIM_TIM8_ETR_ADC3_AWD3

TIM_TIM20_ETR_GPIO

TIM_TIM20_ETR_COMP1

TIM_TIM20_ETR_COMP2

TIM_TIM20_ETR_COMP3

TIM_TIM20_ETR_COMP4

TIM_TIM20_ETR_COMP5

TIM_TIM20_ETR_COMP6

TIM_TIM20_ETR_COMP7

TIM_TIM20_ETR_ADC3_AWD1

TIM_TIM20_ETR_ADC3_AWD2

TIM_TIM20_ETR_ADC3_AWD3

TIM_TIM20_ETR_ADC5_AWD1

TIM_TIM20_ETR_ADC5_AWD2

TIM_TIM20_ETR_ADC5_AWD3

TIM Extended Bitfield SMS preload enabling

TIM_SMS_PRELOAD_SOURCE_UPDATE

Prelaod of SMS bitfield is disabled

TIM_SMS_PRELOAD_SOURCE_INDEX

Preload of SMS bitfield is enabled

TIM Extended Timer input selection

TIM_TIM1_TI1_GPIO

TIM1 input 1 is connected to GPIO

TIM_TIM1_TI1_COMP1

TIM1 input 1 is connected to COMP1_OUT

TIM_TIM1_TI1_COMP2

TIM1 input 1 is connected to COMP2_OUT

TIM_TIM1_TI1_COMP3

TIM1 input 1 is connected to COMP3_OUT

TIM_TIM1_TI1_COMP4

TIM1 input 1 is connected to COMP4_OUT

TIM_TIM2_TI1_GPIO

TIM2 input 1 is connected to GPIO

TIM_TIM2_TI1_COMP1

TIM2 input 1 is connected to COMP1_OUT

TIM_TIM2_TI1_COMP2

TIM2 input 1 is connected to COMP2_OUT

TIM_TIM2_TI1_COMP3

TIM2 input 1 is connected to COMP3_OUT

TIM_TIM2_TI1_COMP4

TIM2 input 1 is connected to COMP4_OUT

TIM_TIM2_TI1_COMP5

TIM2 input 1 is connected to COMP5_OUT

TIM_TIM2_TI2_GPIO

TIM2 input 2 is connected to GPIO

TIM_TIM2_TI2_COMP1

TIM2 input 2 is connected to COMP1_OUT

TIM_TIM2_TI2_COMP2

TIM2 input 2 is connected to COMP2_OUT

TIM_TIM2_TI2_COMP3

TIM2 input 2 is connected to COMP3_OUT

TIM_TIM2_TI2_COMP4

TIM2 input 2 is connected to COMP4_OUT

TIM_TIM2_TI2_COMP6

TIM2 input 2 is connected to COMP6_OUT

TIM_TIM2_TI3_GPIO

TIM2 input 3 is connected to GPIO

TIM_TIM2_TI3_COMP4

TIM2 input 3 is connected to COMP4_OUT

TIM_TIM2_TI4_GPIO

TIM2 input 4 is connected to GPIO

TIM_TIM2_TI4_COMP1

TIM2 input 4 is connected to COMP1_OUT

TIM_TIM2_TI4_COMP2

TIM2 input 4 is connected to COMP2_OUT

TIM_TIM3_TI1_GPIO

TIM3 input 1 is connected to GPIO

TIM_TIM3_TI1_COMP1

TIM3 input 1 is connected to COMP1_OUT

TIM_TIM3_TI1_COMP2

TIM3 input 1 is connected to COMP2_OUT

TIM_TIM3_TI1_COMP3

TIM3 input 1 is connected to COMP3_OUT

TIM_TIM3_TI1_COMP4

TIM3 input 1 is connected to COMP4_OUT

TIM_TIM3_TI1_COMP5

TIM3 input 1 is connected to COMP5_OUT

TIM_TIM3_TI1_COMP6

TIM3 input 1 is connected to COMP6_OUT

TIM_TIM3_TI1_COMP7

TIM3 input 1 is connected to COMP7_OUT

TIM_TIM3_TI2_GPIO

TIM3 input 2 is connected to GPIO

TIM_TIM3_TI2_COMP1

TIM3 input 2 is connected to COMP1_OUT

TIM_TIM3_TI2_COMP2

TIM3 input 2 is connected to COMP2_OUT

TIM_TIM3_TI2_COMP3

TIM3 input 2 is connected to COMP3_OUT

TIM_TIM3_TI2_COMP4

TIM3 input 2 is connected to COMP4_OUT

TIM_TIM3_TI2_COMP5

TIM3 input 2 is connected to COMP5_OUT

TIM_TIM3_TI2_COMP6

TIM3 input 2 is connected to COMP6_OUT

TIM_TIM3_TI2_COMP7

TIM3 input 2 is connected to COMP7_OUT

TIM_TIM3_TI3_GPIO

TIM3 input 3 is connected to GPIO

TIM_TIM3_TI3_COMP3

TIM3 input 3 is connected to COMP3_OUT

TIM_TIM4_TI1_GPIO

TIM4 input 1 is connected to GPIO

TIM_TIM4_TI1_COMP1

TIM4 input 1 is connected to COMP1_OUT

TIM_TIM4_TI1_COMP2

TIM4 input 1 is connected to COMP2_OUT

TIM_TIM4_TI1_COMP3

TIM4 input 1 is connected to COMP3_OUT

TIM_TIM4_TI1_COMP4

TIM4 input 1 is connected to COMP4_OUT

TIM_TIM4_TI1_COMP5

TIM4 input 1 is connected to COMP5_OUT

TIM_TIM4_TI1_COMP6

TIM4 input 1 is connected to COMP6_OUT

TIM_TIM4_TI1_COMP7

TIM4 input 1 is connected to COMP7_OUT

TIM_TIM4_TI2_GPIO

TIM4 input 2 is connected to GPIO

TIM_TIM4_TI2_COMP1

TIM4 input 2 is connected to COMP1_OUT

TIM_TIM4_TI2_COMP2

TIM4 input 2 is connected to COMP2_OUT

TIM_TIM4_TI2_COMP3

TIM4 input 2 is connected to COMP3_OUT

TIM_TIM4_TI2_COMP4

TIM4 input 2 is connected to COMP4_OUT

TIM_TIM4_TI2_COMP5

TIM4 input 2 is connected to COMP5_OUT

TIM_TIM4_TI2_COMP6

TIM4 input 2 is connected to COMP6_OUT

TIM_TIM4_TI2_COMP7

TIM4 input 2 is connected to COMP7_OUT

TIM_TIM4_TI3_GPIO

TIM4 input 3 is connected to GPIO

TIM_TIM4_TI3_COMP5

TIM4 input 3 is connected to COMP5_OUT

TIM_TIM4_TI4_GPIO

TIM4 input 4 is connected to GPIO

TIM_TIM4_TI4_COMP6

TIM4 input 4 is connected to COMP6_OUT

TIM_TIM5_TI1_GPIO

TIM5 input 1 is connected to GPIO

TIM_TIM5_TI1_LSI

TIM5 input 1 is connected to LSI

TIM_TIM5_TI1_LSE

TIM5 input 1 is connected to LSE

TIM_TIM5_TI1_RTC_WK

TIM5 input 1 is connected to RTC_WAKEUP

TIM_TIM5_TI1_COMP1

TIM5 input 1 is connected to COMP1_OUT

TIM_TIM5_TI1_COMP2

TIM5 input 1 is connected to COMP2_OUT

TIM_TIM5_TI1_COMP3

TIM5 input 1 is connected to COMP3_OUT

TIM_TIM5_TI1_COMP4

TIM5 input 1 is connected to COMP4_OUT

TIM_TIM5_TI1_COMP5

TIM5 input 1 is connected to COMP5_OUT

TIM_TIM5_TI1_COMP6

TIM5 input 1 is connected to COMP6_OUT

TIM_TIM5_TI1_COMP7

TIM5 input 1 is connected to COMP7_OUT

TIM_TIM5_TI2_GPIO

TIM5 input 2 is connected to GPIO

TIM_TIM5_TI2_COMP1

TIM5 input 2 is connected to COMP1_OUT

TIM_TIM5_TI2_COMP2

TIM5 input 2 is connected to COMP2_OUT

TIM_TIM5_TI2_COMP3

TIM5 input 2 is connected to COMP3_OUT

TIM_TIM5_TI2_COMP4

TIM5 input 2 is connected to COMP4_OUT

TIM_TIM5_TI2_COMP5

TIM5 input 2 is connected to COMP5_OUT

TIM_TIM5_TI2_COMP6

TIM5 input 2 is connected to COMP6_OUT

TIM_TIM5_TI2_COMP7

TIM5 input 2 is connected to COMP7_OUT

TIM_TIM8_TI1_GPIO

TIM8 input 1 is connected to GPIO

TIM_TIM8_TI1_COMP1

TIM8 input 1 is connected to COMP1_OUT

TIM_TIM8_TI1_COMP2

TIM8 input 1 is connected to COMP2_OUT

TIM_TIM8_TI1_COMP3

TIM8 input 1 is connected to COMP3_OUT

TIM_TIM8_TI1_COMP4

TIM8 input 1 is connected to COMP4_OUT

TIM_TIM15_TI1_GPIO

TIM15 input 1 is connected to GPIO

TIM_TIM15_TI1_LSE

TIM15 input 1 is connected to LSE

TIM_TIM15_TI1_COMP1

TIM15 input 1 is connected to COMP1_OUT

TIM_TIM15_TI1_COMP2

TIM15 input 1 is connected to COMP2_OUT

TIM_TIM15_TI1_COMP5

TIM15 input 1 is connected to COMP5_OUT

TIM_TIM15_TI1_COMP7

TIM15 input 1 is connected to COMP7_OUT

TIM_TIM15_TI2_GPIO

TIM15 input 2 is connected to GPIO

TIM_TIM15_TI2_COMP2

TIM15 input 2 is connected to COMP2_OUT

TIM_TIM15_TI2_COMP3

TIM15 input 2 is connected to COMP3_OUT

TIM_TIM15_TI2_COMP6

TIM15 input 2 is connected to COMP6_OUT

TIM_TIM15_TI2_COMP7

TIM15 input 2 is connected to COMP7_OUT

TIM_TIM16_TI1_GPIO

TIM16 input 1 is connected to GPIO

TIM_TIM16_TI1_COMP6

TIM16 input 1 is connected to COMP6_OUT

TIM_TIM16_TI1_MCO

TIM16 input 1 is connected to MCO

TIM_TIM16_TI1_HSE_32

TIM16 input 1 is connected to HSE/32

TIM_TIM16_TI1_RTC_WK

TIM16 input 1 is connected to RTC_WAKEUP

TIM_TIM16_TI1_LSE

TIM16 input 1 is connected to LSE

TIM_TIM16_TI1_LSI

TIM16 input 1 is connected to LSI

TIM_TIM17_TI1_GPIO

TIM17 input 1 is connected to GPIO

TIM_TIM17_TI1_COMP5

TIM17 input 1 is connected to COMP5_OUT

TIM_TIM17_TI1_MCO

TIM17 input 1 is connected to MCO

TIM_TIM17_TI1_HSE_32

TIM17 input 1 is connected to HSE/32

TIM_TIM17_TI1_RTC_WK

TIM17 input 1 is connected to RTC_WAKEUP

TIM_TIM17_TI1_LSE

TIM17 input 1 is connected to LSE

TIM_TIM17_TI1_LSI

TIM17 input 1 is connected to LSI

TIM_TIM20_TI1_GPIO

TIM20 input 1 is connected to GPIO

TIM_TIM20_TI1_COMP1

TIM20 input 1 is connected to COMP1_OUT

TIM_TIM20_TI1_COMP2

TIM20 input 1 is connected to COMP2_OUT

TIM_TIM20_TI1_COMP3

TIM20 input 1 is connected to COMP3_OUT

TIM_TIM20_TI1_COMP4

TIM20 input 1 is connected to COMP4_OUT

60 HAL UART Generic Driver

60.1 UART Firmware driver registers structures

60.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32g4xx_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***

  This member configures the UART communication baud rate. The baud rate register is computed using the following formula: LPUART: ======= Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where lpuart_ker_ck_pres is the UART input clock divided by a prescaler UART: =====

  – If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))

  – If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where uart_ker_ck_pres is the UART input clock divided by a prescaler

- ***uint32_t UART_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.

- ***uint32_t UART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.

- ***uint32_t UART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***UART_Parity***

  Note:

  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).

- ***uint32_t UART_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.

- ***uint32_t UART_InitTypeDef::HwFlowCtl***

  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.

- ***uint32_t UART_InitTypeDef::OverSampling***

  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.

- ***uint32_t UART_InitTypeDef::OneBitSampling***

  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.

- ***uint32_t UART_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

60.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32g4xx_hal_uart.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***

  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***.
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***

  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***

  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***

  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***

  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***.
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***

  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***

  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***

  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***

  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***.
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***

  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***.

60.1.3 __UART_HandleTypeDef

__UART_HandleTypeDef is defined in the stm32g4xx_hal_uart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***UART_AdvFeatureInitTypeDef AdvancedInit***

- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint32_t FifoMode***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef gState***
- ***__IO HAL_UART_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***UART_AdvFeatureInitTypeDef __UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***uint8_t* __UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::TxXferCount***
  UART Tx Transfer Counter
- ***uint8_t* __UART_HandleTypeDef::pRxBuffPtr***
  Pointer to UART Rx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::RxXferSize***
  UART Rx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***uint16_t __UART_HandleTypeDef::Mask***
  UART Rx RDR register mask
- ***uint32_t __UART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***UARTEx_FIFO_mode***.
- ***uint16_t __UART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __UART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***void(* __UART_HandleTypeDef::RxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __UART_HandleTypeDef::TxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler

- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code

60.2 UART Firmware driver API description

The following section lists the various functions of the UART library.

60.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   – Enable the USARTx interface clock.
   – UART pins configuration:
     ◦ Enable the clock for the UART GPIOs.
     ◦ Configure these UART pins as alternate function pull-up.
   – NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
     ◦ Configure the USARTx interrupt priority.
     ◦ Enable the NVIC USART IRQ handle.
   – UART interrupts handling:

Note: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt, RX/TX FIFOs related interrupts and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.

   – DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
     ◦ Declare a DMA handle structure for the Tx/Rx channel.
     ◦ Enable the DMAx interface clock.
     ◦ Configure the declared DMA handle structure with the required Tx/Rx parameters.
     ◦ Configure the DMA Tx/Rx channel.
     ◦ Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Prescaler value , Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.

8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note: These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

60.2.2 Callback registration

The compilation define USE_HAL_UART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_UART_RegisterCallback() to register a user callback. Function @ref HAL_UART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_UART_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_UART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit.

By default, after the @ref HAL_UART_Init() and when the state is HAL_UART_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples @ref HAL_UART_TxCpltCallback(), @ref HAL_UART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_UART_Init() and @ref HAL_UART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_UART_Init() and @ref HAL_UART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_UART_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_UART_STATE_READY or HAL_UART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_UART_RegisterCallback() before calling @ref HAL_UART_DeInit() or @ref HAL_UART_Init() function.

When The compilation define USE_HAL_UART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

60.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init()and HAL_MultiProcessor_Init()API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_UART_Init***
- ***HAL_HalfDuplex_Init***
- ***HAL_LIN_Init***
- ***HAL_MultiProcessor_Init***
- ***HAL_UART_DeInit***
- ***HAL_UART_MspInit***
- ***HAL_UART_MspDeInit***

60.2.4 IO operation functions

This section contains the following APIs:

- ***HAL_UART_Transmit***
- ***HAL_UART_Receive***
- ***HAL_UART_Transmit_IT***
- ***HAL_UART_Receive_IT***
- ***HAL_UART_Transmit_DMA***
- ***HAL_UART_Receive_DMA***
- ***HAL_UART_DMAPause***
- ***HAL_UART_DMAResume***
- ***HAL_UART_DMAStop***

- *HAL_UART_Abort*
- *HAL_UART_AbortTransmit*
- *HAL_UART_AbortReceive*
- *HAL_UART_Abort_IT*
- *HAL_UART_AbortTransmit_IT*
- *HAL_UART_AbortReceive_IT*
- *HAL_UART_IRQHandler*
- *HAL_UART_TxCpltCallback*
- *HAL_UART_TxHalfCpltCallback*
- *HAL_UART_RxCpltCallback*
- *HAL_UART_RxHalfCpltCallback*
- *HAL_UART_ErrorCallback*
- *HAL_UART_AbortCpltCallback*
- *HAL_UART_AbortTransmitCpltCallback*
- *HAL_UART_AbortReceiveCpltCallback*

60.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART.

- HAL_UART_ReceiverTimeout_Config() API allows to configure the receiver timeout value on the fly
- HAL_UART_EnableReceiverTimeout() API enables the receiver timeout feature
- HAL_UART_DisableReceiverTimeout() API disables the receiver timeout feature
- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:

- *HAL_UART_ReceiverTimeout_Config*
- *HAL_UART_EnableReceiverTimeout*
- *HAL_UART_DisableReceiverTimeout*
- *HAL_MultiProcessor_EnableMuteMode*
- *HAL_MultiProcessor_DisableMuteMode*
- *HAL_MultiProcessor_EnterMuteMode*
- *HAL_HalfDuplex_EnableTransmitter*
- *HAL_HalfDuplex_EnableReceiver*
- *HAL_LIN_SendBreak*

60.2.6 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:

- *HAL_UART_GetState*
- *HAL_UART_GetError*

60.2.7 Detailed description of functions

HAL_UART_Init

Function name

HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)

Function description

Initialize the UART mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_Init

Function name

HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)

Function description

Initialize the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_LIN_Init

Function name

HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function description

Initialize the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **BreakDetectLength:** Specifies the LIN break detection length. This parameter can be one of the following values:
  - UART_LINBREAKDETECTLENGTH_10B 10-bit break detection
  - UART_LINBREAKDETECTLENGTH_11B 11-bit break detection

Return values

- **HAL:** status

HAL_MultiProcessor_Init

Function name

HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)

Function description

Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.
- **Address:** UART node address (4-, 6-, 7- or 8-bit long).
- **WakeUpMethod:** Specifies the UART wakeup method. This parameter can be one of the following values:
  - UART_WAKEUPMETHOD_IDLELINE WakeUp by an idle line detection
  - UART_WAKEUPMETHOD_ADDRESSMARK WakeUp by an address mark

Return values

- **HAL:** status

Notes

- If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.
- If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

HAL_UART_DeInit

Function name

HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_MspInit

Function name

void HAL_UART_MspInit (UART_HandleTypeDef * huart)

Function description

Initialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_MspDeInit

Function name

void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.
- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_UART_Receive

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_IT

Function name

HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_DMAPause

Function name

HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)

Function description

Pause the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAResume

Function name

HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)

Function description

Resume the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAStop

Function name

HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)

Function description

Stop the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_Abort

Function name

HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name

HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name

void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)

Function description

Handle UART interrupt request.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxHalfCpltCallback

Function name

void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxCpltCallback

Function name

void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxHalfCpltCallback

Function name

void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxCpltCallback

Function name

void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_ErrorCallback

Function name

void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)

Function description

UART error callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortCpltCallback

Function name

void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortTransmitCpltCallback

Function name

void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortReceiveCpltCallback

Function name

void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Receive Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_ReceiverTimeout_Config

Function name

void HAL_UART_ReceiverTimeout_Config (UART_HandleTypeDef * huart, uint32_t TimeoutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **TimeoutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.

Return values

- **None:**

HAL_UART_EnableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_EnableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_UART_DisableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_DisableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Disable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

Transmit break characters.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** UART handle.

Return values

- **None:**

Notes

- To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

Enable the UART transmitter and disable the UART receiver.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver and disable the UART transmitter.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status.

HAL_UART_GetState

Function name

HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)

Function description

Return the UART handle state.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **HAL:** state

HAL_UART_GetError

Function name

uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)

Function description

Return the UART handle error code.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **UART:** Error Code

UART_SetConfig

Function name

HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)

Function description

Initialize the callbacks to their default values.

Parameters

- **huart:** UART handle.
- **huart:** UART handle.

Return values

- **none:** Configure the UART peripheral.
- **HAL:** status

UART_CheckIdleState

Function name

HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)

Function description

Check the UART Idle State.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name

HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)

Function description

Handle UART Communication Timeout.

Parameters

- **huart:** UART handle.
- **Flag:** Specifies the UART flag to check
- **Status:** Flag status (SET or RESET)
- **Tickstart:** Tick start value
- **Timeout:** Timeout duration

Return values

- **HAL:** status

UART_AdvFeatureConfig

Function name

void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral advanced features.

Parameters

- **huart:** UART handle.

Return values

- **None:**

60.3 UART Firmware driver defines

The following section lists the various define and macros of the module.

60.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

No advanced feature initialization

UART_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

UART_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

UART_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

UART_ADVFEATURE_SWAP_INIT

TX/RX pins swap

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

UART_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

UART_ADVFEATURE_AUTOBAUDRATE_INIT

Auto Baud rate detection initialization

UART_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT

Auto Baud rate detection on start bit

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

Auto Baud rate detection on falling edge

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME

Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

Auto Baud rate detection on 0x55 frame detection

UART Clock Prescaler

UART_PRESCALER_DIV1

fclk_pres = fclk

UART_PRESCALER_DIV2

fclk_pres = fclk/2

UART_PRESCALER_DIV4

fclk_pres = fclk/4

UART_PRESCALER_DIV6

fclk_pres = fclk/6

UART_PRESCALER_DIV8

fclk_pres = fclk/8

UART_PRESCALER_DIV10

fclk_pres = fclk/10

UART_PRESCALER_DIV12

fclk_pres = fclk/12

UART_PRESCALER_DIV16

fclk_pres = fclk/16

UART_PRESCALER_DIV32

fclk_pres = fclk/32

UART_PRESCALER_DIV64

fclk_pres = fclk/64

UART_PRESCALER_DIV128

fclk_pres = fclk/128

UART_PRESCALER_DIV256

fclk_pres = fclk/256

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS

UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS

UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS

UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

UART_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

UART_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE

UART DMA RX disabled

UART_DMA_RX_ENABLE

UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE

UART DMA TX disabled

UART_DMA_TX_ENABLE

UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH

Driver enable signal is active high

UART_DE_POLARITY_LOW

Driver enable signal is active low

UART Error Definition

HAL_UART_ERROR_NONE

No error

HAL_UART_ERROR_PE

Parity error

HAL_UART_ERROR_NE

Noise error

HAL_UART_ERROR_FE

Frame error

HAL_UART_ERROR_ORE

Overrun error

HAL_UART_ERROR_DMA

DMA transfer error

HAL_UART_ERROR_RTO

Receiver Timeout error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_RTOF Receiver Timeout clear flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_TXFECF

Description:

- Clear the UART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_GET_FLAG

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_TXFT TXFIFO threshold flag
  - UART_FLAG_RXFT RXFIFO threshold flag
  - UART_FLAG_RXFF RXFIFO Full flag
  - UART_FLAG_TXFE TXFIFO Empty flag
  - UART_FLAG_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag
  - UART_FLAG_WUF Wake up from stop mode flag
  - UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
  - UART_FLAG_SBKF Send Break flag
  - UART_FLAG_CMF Character match flag
  - UART_FLAG_BUSY Busy flag
  - UART_FLAG_ABRF Auto Baud rate detection flag
  - UART_FLAG_ABRE Auto Baud rate detection error flag
  - UART_FLAG_CTS CTS Change flag
  - UART_FLAG_LBDF LIN Break detection flag
  - UART_FLAG_TXE Transmit data register empty flag
  - UART_FLAG_TXFNF UART TXFIFO not full flag
  - UART_FLAG_TC Transmission Complete flag
  - UART_FLAG_RXNE Receive data register not empty flag
  - UART_FLAG_RXFNE UART RXFIFO not empty flag
  - UART_FLAG_RTOF Receiver Timeout flag
  - UART_FLAG_IDLE Idle Line detection flag
  - UART_FLAG_ORE Overrun Error flag
  - UART_FLAG_NE Noise Error flag
  - UART_FLAG_FE Framing Error flag
  - UART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_GET_IT

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_GET_IT_SOURCE

Description:

- Check whether the specified UART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_CLEAR_IT

Description:

- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_RTOF Receiver timeout clear flag
  - UART_CLEAR_TXFECF TXFIFO empty Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - UART_SENDBREAK_REQUEST Send Break Request
  - UART_MUTE_MODE_REQUEST Mute Mode Request
  - UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_DISABLE

Description:

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

UART Status Flags

UART_FLAG_TXFT

UART TXFIFO threshold flag

UART_FLAG_RXFT

UART RXFIFO threshold flag

UART_FLAG_RXFF

UART RXFIFO Full flag

UART_FLAG_TXFE

UART TXFIFO Empty flag

UART_FLAG_REACK

UART receive enable acknowledge flag

UART_FLAG_TEACK

UART transmit enable acknowledge flag

UART_FLAG_WUF

UART wake-up from stop mode flag

UART_FLAG_RWU

UART receiver wake-up from mute mode flag

UART_FLAG_SBKF

UART send break flag

UART_FLAG_CMF

UART character match flag

UART_FLAG_BUSY

UART busy flag

UART_FLAG_ABRF

UART auto Baud rate flag

UART_FLAG_ABRE

UART auto Baud rate error

UART_FLAG_RTOF

UART receiver timeout flag

UART_FLAG_CTS

UART clear to send flag

UART_FLAG_CTSIF

UART clear to send interrupt flag

UART_FLAG_LBDF

UART LIN break detection flag

UART_FLAG_TXE

UART transmit data register empty

UART_FLAG_TXFNF

UART TXFIFO not full

UART_FLAG_TC

UART transmission complete

UART_FLAG_RXNE

UART read data register not empty

UART_FLAG_RXFNE

UART RXFIFO not empty

UART_FLAG_IDLE

UART idle flag

UART_FLAG_ORE

UART overrun error

UART_FLAG_NE

UART noise error

UART_FLAG_FE

UART frame error

UART_FLAG_PE

UART parity error

UART Half Duplex Selection

UART_HALF_DUPLEX_DISABLE

UART half-duplex disabled

UART_HALF_DUPLEX_ENABLE

UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE

No hardware control

UART_HWCONTROL_RTS

Request To Send

UART_HWCONTROL_CTS

Clear To Send

UART_HWCONTROL_RTS_CTS

Request and Clear To Send

UART Interruptions Flag Mask

UART_IT_MASK

UART interruptions flags mask

UART Interrupts Definition

UART_IT_PE

UART parity error interruption

UART_IT_TXE

UART transmit data register empty interruption

UART_IT_TXFNF

UART TX FIFO not full interruption

UART_IT_TC

UART transmission complete interruption

UART_IT_RXNE

UART read data register not empty interruption

UART_IT_RXFNE

UART RXFIFO not empty interruption

UART_IT_IDLE

UART idle interruption

UART_IT_LBD

UART LIN break detection interruption

UART_IT_CTS

UART CTS interruption

UART_IT_CM

UART character match interruption

UART_IT_WUF

UART wake-up from stop mode interruption

UART_IT_RXFF

UART RXFIFO full interruption

UART_IT_TXFE

UART TXFIFO empty interruption

UART_IT_RXFT

UART RXFIFO threshold reached interruption

UART_IT_TXFT

UART TXFIFO threshold reached interruption

UART_IT_RTO

UART receiver timeout interruption

UART_IT_ERR

UART error interruption

UART_IT_ORE

UART overrun error interruption

UART_IT_NE

UART noise error interruption

UART_IT_FE

UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF

Parity Error Clear Flag

UART_CLEAR_FEF

Framing Error Clear Flag

UART_CLEAR_NEF

Noise Error detected Clear Flag

UART_CLEAR_OREF

Overrun Error Clear Flag

UART_CLEAR_IDLEF

IDLE line detected Clear Flag

UART_CLEAR_TXFECF

TXFIFO empty clear flag

UART_CLEAR_TCF

Transmission Complete Clear Flag

UART_CLEAR_LBDF

LIN Break Detection Clear Flag

UART_CLEAR_CTSF

CTS Interrupt Clear Flag

UART_CLEAR_CMF

Character Match Clear Flag

UART_CLEAR_WUF

Wake Up from stop mode Clear Flag

UART_CLEAR_RTOF

UART receiver timeout clear flag

UART Local Interconnection Network mode

UART_LIN_DISABLE

Local Interconnect Network disable

UART_LIN_ENABLE

Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B

LIN 10-bit break detection length

UART_LINBREAKDETECTLENGTH_11B

LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX

RX mode

UART_MODE_TX

TX mode

UART_MODE_TX_RX

RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

UART_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE

UART mute mode disable

UART_ADVFEATURE_MUTEMODE_ENABLE

UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disable

UART_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

UART_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16

Oversampling by 16

UART_OVERSAMPLING_8

Oversampling by 8

UART Parity

UART_PARITY_NONE

No parity

UART_PARITY_EVEN

Even parity

UART_PARITY_ODD

Odd parity

UART Receiver Timeout

UART_RECEIVER_TIMEOUT_DISABLE

UART Receiver Timeout disable

UART_RECEIVER_TIMEOUT_ENABLE

UART Receiver Timeout enable

UART Request Parameters

UART_AUTOBAUD_REQUEST

Auto-Baud Rate Request

UART_SENDBREAK_REQUEST

Send Break Request

UART_MUTE_MODE_REQUEST

Mute Mode Request

UART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

UART_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

UART_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

UART State

UART_STATE_DISABLE

UART disabled

UART_STATE_ENABLE

UART enabled

UART State Code Definition

HAL_UART_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_UART_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_UART_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_UART_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_UART_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_UART_STATE_ERROR

Error Value is allowed for gState only

UART Number of Stop Bits

UART_STOPBITS_0_5

UART frame with 0.5 stop bit

UART_STOPBITS_1

UART frame with 1 stop bit

UART_STOPBITS_1_5

UART frame with 1.5 stop bits

UART_STOPBITS_2

UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE

UART stop mode disable

UART_ADVFEATURE_STOPMODE_ENABLE

UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE

UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

UART_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS

UART wake-up on address

UART_WAKEUP_ON_STARTBIT

UART wake-up on start bit

UART_WAKEUP_ON_READDATA_NONEMPTY

UART wake-up on receive data register not empty or RXFIFO is not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE

UART wake-up on idle line

UART_WAKEUPMETHOD_ADDRESSMARK

UART wake-up on address mark

61 HAL UART Extension Driver

61.1 UARTEx Firmware driver registers structures

61.1.1 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32g4xx_hal_uart_ex.h

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***

  Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***

  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***

  UART/USART node address (7-bit long max).

61.2 UARTEx Firmware driver API description

The following section lists the various functions of the UARTEx library.

61.2.1 UART peripheral extended features

61.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- *HAL_RS485Ex_Init*

61.2.3 IO operation functions

This section contains the following APIs:

- *HAL_UARTEx_WakeupCallback*
- *HAL_UARTEx_RxFifoFullCallback*
- *HAL_UARTEx_TxFifoEmptyCallback*

61.2.4 Peripheral Control functions

This section provides the following functions:

- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_UARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_UARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_UARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- *HAL_MultiProcessorEx_AddressLength_Set*
- *HAL_UARTEx_StopModeWakeUpSourceConfig*
- *HAL_UARTEx_EnableStopMode*
- *HAL_UARTEx_DisableStopMode*
- *HAL_UARTEx_EnableFifoMode*
- *HAL_UARTEx_DisableFifoMode*
- *HAL_UARTEx_SetTxFifoThreshold*
- *HAL_UARTEx_SetRxFifoThreshold*

61.2.5 Detailed description of functions

HAL_RS485Ex_Init

Function name

HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)

Function description

Initialize the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **Polarity:** Select the driver enable polarity. This parameter can be one of the following values:
  - UART_DE_POLARITY_HIGH DE signal is active high
  - UART_DE_POLARITY_LOW DE signal is active low
- **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)
- **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

Return values

- **HAL:** status

HAL_UARTEx_WakeupCallback

Function name

void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)

Function description

UART wakeup from Stop mode callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_RxFifoFullCallback

Function name

void HAL_UARTEx_RxFifoFullCallback (UART_HandleTypeDef * huart)

Function description

UART RX Fifo full callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_TxFifoEmptyCallback

Function name

void HAL_UARTEx_TxFifoEmptyCallback (UART_HandleTypeDef * huart)

Function description

UART TX Fifo empty callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name

HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)

Function description

Set Wakeup from Stop mode interrupt flag selection.

Parameters

- **huart:** UART handle.
- **WakeUpSelection:** Address match, Start Bit detection or RXNE/RXFNE bit status. This parameter can be one of the following values:
  - UART_WAKEUP_ON_ADDRESS
  - UART_WAKEUP_ON_STARTBIT
  - UART_WAKEUP_ON_READDATA_NONEMPTY

Return values

- **HAL:** status

Notes

- It is the application responsibility to enable the interrupt used as usart_wkup interrupt source before entering low-power mode.

HAL_UARTEx_EnableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)

Function description

Enable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- The UART is able to wake up the MCU from Stop 1 mode as long as UART clock is HSI or LSE.

HAL_UARTEx_DisableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)

Function description

Disable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessorEx_AddressLength_Set

Function name

HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)

Function description

By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).

Parameters

- **huart:** UART handle.
- **AddressLength:** This parameter can be one of the following values:
  - UART_ADDRESS_DETECT_4B 4-bit long address
  - UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address

Return values

- **HAL:** status

Notes

- Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableFifoMode (UART_HandleTypeDef * huart)

Function description

Enable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableFifoMode (UART_HandleTypeDef * huart)

Function description

Disable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetTxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - UART_TXFIFO_THRESHOLD_1_8
  - UART_TXFIFO_THRESHOLD_1_4
  - UART_TXFIFO_THRESHOLD_1_2
  - UART_TXFIFO_THRESHOLD_3_4
  - UART_TXFIFO_THRESHOLD_7_8
  - UART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetRxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - UART_RXFIFO_THRESHOLD_1_8
  - UART_RXFIFO_THRESHOLD_1_4
  - UART_RXFIFO_THRESHOLD_1_2
  - UART_RXFIFO_THRESHOLD_3_4
  - UART_RXFIFO_THRESHOLD_7_8
  - UART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

61.3 UARTEx Firmware driver defines

The following section lists the various define and macros of the module.

61.3.1 UARTEx

UARTEx

UARTEx FIFO mode

UART_FIFOMODE_DISABLE

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

UARTEx RXFIFO threshold level

UART_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

UART_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

UART_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

UART_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

UART_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

UART_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

UARTEx TXFIFO threshold level

UART_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

UART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

UART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

UART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

UART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

UART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

UARTEx WakeUp Address Length

UART_ADDRESS_DETECT_4B

4-bit long wake-up address

UART_ADDRESS_DETECT_7B

7-bit long wake-up address

UARTEx Word Length

UART_WORDLENGTH_7B

7-bit long UART frame

UART_WORDLENGTH_8B

8-bit long UART frame

UART_WORDLENGTH_9B

9-bit long UART frame

62 HAL USART Generic Driver

62.1 USART Firmware driver registers structures

62.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32g4xx_hal_usart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***

  This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register[15:4] = ((2 * fclk_pres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * fclk_pres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where fclk_pres is the USART input clock frequency (fclk) divided by a prescaler.

  Note:

  – Oversampling by 8 is systematically applied to achieve high baud rates.

- ***uint32_t USART_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***.

- ***uint32_t USART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***.

- ***uint32_t USART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***USART_Parity***

  Note:

  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).

- ***uint32_t USART_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***.

- ***uint32_t USART_InitTypeDef::CLKPolarity***

  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***.

- ***uint32_t USART_InitTypeDef::CLKPhase***

  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***.

- ***uint32_t USART_InitTypeDef::CLKLastBit***

  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***.

- ***uint32_t USART_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***USART_ClockPrescaler***.

62.1.2 __USART_HandleTypeDef

__USART_HandleTypeDef is defined in the stm32g4xx_hal_usart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***USART_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t SlaveMode***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_USART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __USART_HandleTypeDef::Instance***
  USART registers base address
- ***USART_InitTypeDef __USART_HandleTypeDef::Init***
  USART communication parameters
- ***uint8_t* __USART_HandleTypeDef::pTxBuffPtr***
  Pointer to USART Tx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::TxXferSize***
  USART Tx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::TxXferCount***
  USART Tx Transfer Counter
- ***uint8_t* __USART_HandleTypeDef::pRxBuffPtr***
  Pointer to USART Rx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::RxXferSize***
  USART Rx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t __USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***uint16_t __USART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __USART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __USART_HandleTypeDef::SlaveMode***
  Enable/Disable UART SPI Slave Mode. This parameter can be a value of ***USARTEx_Slave_Mode***

- ***uint32_t __USART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***USARTEx_FIFO_mode***.
- ***void(* __USART_HandleTypeDef::RxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Rx IRQ handler
- ***void(* __USART_HandleTypeDef::TxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef __USART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_USART_StateTypeDef __USART_HandleTypeDef::State***
  USART communication state
- ***__IO uint32_t __USART_HandleTypeDef::ErrorCode***
  USART Error code

62.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

62.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   – Enable the USARTx interface clock.
   – USART pins configuration:
     ◦ Enable the clock for the USART GPIOs.
     ◦ Configure these USART pins as alternate function pull-up.
   – NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
     ◦ Configure the USARTx interrupt priority.
     ◦ Enable the NVIC USART IRQ handle.
   – USART interrupts handling:

Note: The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.

   – DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
     ◦ Declare a DMA handle structure for the Tx/Rx channel.
     ◦ Enable the DMAx interface clock.
     ◦ Configure the declared DMA handle structure with the required Tx/Rx parameters.
     ◦ Configure the DMA Tx/Rx channel.
     ◦ Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, and Mode (Receiver/Transmitter) in the husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

Note: To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's HAL_UARTEx_StopModeWakeUpSourceConfig(), HAL_UARTEx_EnableStopMode() and HAL_UARTEx_DisableStopMode() in casting the USART handle to UART type UART_HandleTypeDef.

62.2.2 Callback registration

The compilation define USE_HAL_USART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function @ref HAL_USART_RegisterCallback() to register a user callback. Function @ref HAL_USART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_USART_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_USART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit.

By default, after the @ref HAL_USART_Init() and when the state is HAL_USART_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples @ref HAL_USART_TxCpltCallback(), @ref HAL_USART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_USART_Init() and @ref HAL_USART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_USART_Init() and @ref HAL_USART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_USART_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_USART_STATE_READY or HAL_USART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_USART_RegisterCallback() before calling @ref HAL_USART_DeInit() or @ref HAL_USART_Init() function.

When The compilation define USE_HAL_USART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

62.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- ***HAL_USART_Init***
- ***HAL_USART_DeInit***
- ***HAL_USART_MspInit***
- ***HAL_USART_MspDeInit***

62.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected
2. Blocking mode API's are :
   - HAL_USART_Transmit() in simplex mode
   - HAL_USART_Receive() in full duplex receive only
   - HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
   - HAL_USART_Transmit_IT() in simplex mode
   - HAL_USART_Receive_IT() in full duplex receive only
   - HAL_USART_TransmitReceive_IT() in full duplex mode
   - HAL_USART_IRQHandler()
4. No-Blocking mode API's with DMA are :
   - HAL_USART_Transmit_DMA() in simplex mode
   - HAL_USART_Receive_DMA() in full duplex receive only
   - HAL_USART_TransmitReceive_DMA() in full duplex mode
   - HAL_USART_DMAPause()
   - HAL_USART_DMAResume()
   - HAL_USART_DMAStop()

5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
   - HAL_USART_TxCpltCallback()
   - HAL_USART_RxCpltCallback()
   - HAL_USART_TxHalfCpltCallback()
   - HAL_USART_RxHalfCpltCallback()
   - HAL_USART_ErrorCallback()
   - HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   - HAL_USART_Abort()
   - HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided:
   - HAL_USART_AbortCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   - Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_USART_Transmit***
- ***HAL_USART_Receive***
- ***HAL_USART_TransmitReceive***
- ***HAL_USART_Transmit_IT***
- ***HAL_USART_Receive_IT***
- ***HAL_USART_TransmitReceive_IT***
- ***HAL_USART_Transmit_DMA***
- ***HAL_USART_Receive_DMA***
- ***HAL_USART_TransmitReceive_DMA***
- ***HAL_USART_DMAPause***
- ***HAL_USART_DMAResume***
- ***HAL_USART_DMAStop***
- ***HAL_USART_Abort***
- ***HAL_USART_Abort_IT***
- ***HAL_USART_IRQHandler***
- ***HAL_USART_TxCpltCallback***
- ***HAL_USART_TxHalfCpltCallback***
- ***HAL_USART_RxCpltCallback***
- ***HAL_USART_RxHalfCpltCallback***
- ***HAL_USART_TxRxCpltCallback***
- ***HAL_USART_ErrorCallback***
- ***HAL_USART_AbortCpltCallback***

62.2.5 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:

- *HAL_USART_GetState*
- *HAL_USART_GetError*

62.2.6 Detailed description of functions

HAL_USART_Init

Function name

HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)

Function description

Initialize the USART mode according to the specified parameters in the USART_InitTypeDef and initialize the associated handle.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DeInit

Function name

HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART peripheral.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_MspInit

Function name

void HAL_USART_MspInit (USART_HandleTypeDef * husart)

Function description

Initialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_MspDeInit

Function name

void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_Transmit

Function name

HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function description

Simplex send an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive

Function name

HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pRxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Full-Duplex Send and Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_IT

Function name

HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Send and Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received/sent.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_DMAPause

Function name

HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)

Function description

Pause the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAResume

Function name

HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)

Function description

Resume the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAStop

Function name

HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)

Function description

Stop the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_Abort

Function name

HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name

HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name

void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function description

Handle USART interrupt request.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxHalfCpltCallback

Function name

void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxCpltCallback

Function name

void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxCpltCallback

Function name

void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxHalfCpltCallback

Function name

void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxRxCpltCallback

Function name

void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx/Rx Transfers completed callback for the non-blocking process.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_ErrorCallback

Function name

void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)

Function description

USART error callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_AbortCpltCallback

Function name

void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)

Function description

USART Abort Complete callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_GetState

Function name

HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)

Function description

Return the USART handle state.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle state

HAL_USART_GetError

Function name

uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)

Function description

Return the USART error code.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle Error Code

62.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

62.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE

USART clock disable

USART_CLOCK_ENABLE

USART clock enable

USART Clock Prescaler

USART_PRESCALER_DIV1

fclk_pres = fclk

USART_PRESCALER_DIV2

fclk_pres = fclk/2

USART_PRESCALER_DIV4

fclk_pres = fclk/4

USART_PRESCALER_DIV6

fclk_pres = fclk/6

USART_PRESCALER_DIV8

fclk_pres = fclk/8

USART_PRESCALER_DIV10

fclk_pres = fclk/10

USART_PRESCALER_DIV12

fclk_pres = fclk/12

USART_PRESCALER_DIV16

fclk_pres = fclk/16

USART_PRESCALER_DIV32

fclk_pres = fclk/32

USART_PRESCALER_DIV64

fclk_pres = fclk/64

USART_PRESCALER_DIV128

fclk_pres = fclk/128

USART_PRESCALER_DIV256

fclk_pres = fclk/256

USART Clock Phase

USART_PHASE_1EDGE

USART frame phase on first clock transition

USART_PHASE_2EDGE

USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW

Driver enable signal is active high

USART_POLARITY_HIGH

Driver enable signal is active low

USART Error Definition

HAL_USART_ERROR_NONE

No error

HAL_USART_ERROR_PE

Parity error

HAL_USART_ERROR_NE

Noise error

HAL_USART_ERROR_FE

Frame error

HAL_USART_ERROR_ORE

Overrun error

HAL_USART_ERROR_DMA

DMA transfer error

HAL_USART_ERROR_UDR

SPI slave underrun error

HAL_USART_ERROR_RTO

Receiver Timeout error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_GET_FLAG

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - USART_FLAG_TXFT TXFIFO threshold flag
  - USART_FLAG_RXFT RXFIFO threshold flag
  - USART_FLAG_RXFF RXFIFO Full flag
  - USART_FLAG_TXFE TXFIFO Empty flag
  - USART_FLAG_REACK Receive enable acknowledge flag
  - USART_FLAG_TEACK Transmit enable acknowledge flag
  - USART_FLAG_BUSY Busy flag
  - USART_FLAG_UDR SPI slave underrun error flag
  - USART_FLAG_TXE Transmit data register empty flag
  - USART_FLAG_TXFNF TXFIFO not full flag
  - USART_FLAG_TC Transmission Complete flag
  - USART_FLAG_RXNE Receive data register not empty flag
  - USART_FLAG_RXFNE RXFIFO not empty flag
  - USART_FLAG_RTOF Receiver Timeout flag
  - USART_FLAG_IDLE Idle Line detection flag
  - USART_FLAG_ORE OverRun Error flag
  - USART_FLAG_NE Noise Error flag
  - USART_FLAG_FE Framing Error flag
  - USART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_USART_CLEAR_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag
  - USART_CLEAR_RTOF Receiver Timeout clear flag
  - USART_CLEAR_UDRF SPI slave underrun error Clear Flag

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear SPI slave underrun error flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_CLEAR_IT

Description:

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_RTOF Receiver timeout clear flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Flags

USART_FLAG_TXFT

USART TXFIFO threshold flag

USART_FLAG_RXFT

USART RXFIFO threshold flag

USART_FLAG_RXFF

USART RXFIFO Full flag

USART_FLAG_TXFE

USART TXFIFO Empty flag

USART_FLAG_REACK

USART receive enable acknowledge flag

USART_FLAG_TEACK

USART transmit enable acknowledge flag

USART_FLAG_BUSY

USART busy flag

USART_FLAG_UDR

SPI slave underrun error flag

USART_FLAG_TXE

USART transmit data register empty

USART_FLAG_TXFNF

USART TXFIFO not full

USART_FLAG_RTOF

USART receiver timeout flag

USART_FLAG_TC

USART transmission complete

USART_FLAG_RXNE

USART read data register not empty

USART_FLAG_RXFNE

USART RXFIFO not empty

USART_FLAG_IDLE

USART idle flag

USART_FLAG_ORE

USART overrun error

USART_FLAG_NE

USART noise error

USART_FLAG_FE

USART frame error

USART_FLAG_PE

USART parity error

USART Interruption Flags Mask

USART_IT_MASK

USART interruptions flags mask

USART_CR_MASK

USART control register mask

USART_CR_POS

USART control register position

USART_ISR_MASK

USART ISR register mask

USART_ISR_POS

USART ISR register position

USART Interrupts Definition

USART_IT_PE

USART parity error interruption

USART_IT_TXE

USART transmit data register empty interruption

USART_IT_TXFNF

USART TX FIFO not full interruption

USART_IT_TC

USART transmission complete interruption

USART_IT_RXNE

USART read data register not empty interruption

USART_IT_RXFNE

USART RXFIFO not empty interruption

USART_IT_IDLE

USART idle interruption

USART_IT_ERR

USART error interruption

USART_IT_ORE

USART overrun error interruption

USART_IT_NE

USART noise error interruption

USART_IT_FE

USART frame error interruption

USART_IT_RXFF

USART RXFIFO full interruption

USART_IT_TXFE

USART TXFIFO empty interruption

USART_IT_RXFT

USART RXFIFO threshold reached interruption

USART_IT_TXFT

USART TXFIFO threshold reached interruption

USART Interruption Clear Flags

USART_CLEAR_PEF

Parity Error Clear Flag

USART_CLEAR_FEF

Framing Error Clear Flag

USART_CLEAR_NEF

Noise Error detected Clear Flag

USART_CLEAR_OREF

OverRun Error Clear Flag

USART_CLEAR_IDLEF

IDLE line detected Clear Flag

USART_CLEAR_TCF

Transmission Complete Clear Flag

USART_CLEAR_UDRF

SPI slave underrun error Clear Flag

USART_CLEAR_TXFECF

TXFIFO Empty Clear Flag

USART_CLEAR_RTOF

USART receiver timeout clear flag

USART Last Bit

USART_LASTBIT_DISABLE

USART frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE

USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX

RX mode

USART_MODE_TX

TX mode

USART_MODE_TX_RX

RX and TX mode

USART Over Sampling

USART_OVERSAMPLING_16

Oversampling by 16

USART_OVERSAMPLING_8

Oversampling by 8

USART Parity

USART_PARITY_NONE

No parity

USART_PARITY_EVEN

Even parity

USART_PARITY_ODD

Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5

USART frame with 0.5 stop bit

USART_STOPBITS_1

USART frame with 1 stop bit

USART_STOPBITS_1_5

USART frame with 1.5 stop bits

USART_STOPBITS_2

USART frame with 2 stop bits

63 HAL USART Extension Driver

63.1 USARTEx Firmware driver API description

The following section lists the various functions of the USARTEx library.

63.1.1 USART peripheral extended features

63.1.2 IO operation functions

This section contains the following APIs:

- ***HAL_USARTEx_RxFifoFullCallback***
- ***HAL_USARTEx_TxFifoEmptyCallback***

63.1.3 Peripheral Control functions

This section provides the following functions:

- HAL_USARTEx_EnableSPISlaveMode() API enables the SPI slave mode
- HAL_USARTEx_DisableSPISlaveMode() API disables the SPI slave mode
- HAL_USARTEx_ConfigNSS API configures the Slave Select input pin (NSS)
- HAL_USARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_USARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_USARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_USARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_USARTEx_EnableSlaveMode***
- ***HAL_USARTEx_DisableSlaveMode***
- ***HAL_USARTEx_ConfigNSS***
- ***HAL_USARTEx_EnableFifoMode***
- ***HAL_USARTEx_DisableFifoMode***
- ***HAL_USARTEx_SetTxFifoThreshold***
- ***HAL_USARTEx_SetRxFifoThreshold***

63.1.4 Detailed description of functions

HAL_USARTEx_RxFifoFullCallback

Function name

void HAL_USARTEx_RxFifoFullCallback (USART_HandleTypeDef * husart)

Function description

USART RX Fifo full callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_TxFifoEmptyCallback

Function name

void HAL_USARTEx_TxFifoEmptyCallback (USART_HandleTypeDef * husart)

Function description

USART TX Fifo empty callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_EnableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableSlaveMode (USART_HandleTypeDef * husart)

Function description

Enable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- When the USART operates in SPI slave mode, it handles data flow using the serial interface clock derived from the external SCLK signal provided by the external master SPI device.
- In SPI slave mode, the USART must be enabled before starting the master communications (or between frames while the clock is stable). Otherwise, if the USART slave is enabled while the master is in the middle of a frame, it will become desynchronized with the master.
- The data register of the slave needs to be ready before the first edge of the communication clock or before the end of the ongoing communication, otherwise the SPI slave will transmit zeros.

HAL_USARTEx_DisableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableSlaveMode (USART_HandleTypeDef * husart)

Function description

Disable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_ConfigNSS

Function name

HAL_StatusTypeDef HAL_USARTEx_ConfigNSS (USART_HandleTypeDef * husart, uint32_t NSSConfig)

Function description

Configure the Slave Select input pin (NSS).

Parameters

- **husart:** USART handle.
- **NSSConfig:** NSS configuration. This parameter can be one of the following values:
  - USART_NSS_HARD
  - USART_NSS_SOFT

Return values

- **HAL:** status

Notes

- Software NSS management: SPI slave will always be selected and NSS input pin will be ignored.
- Hardware NSS management: the SPI slave selection depends on NSS input pin. The slave is selected when NSS is low and deselected when NSS is high.

HAL_USARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableFifoMode (USART_HandleTypeDef * husart)

Function description

Enable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableFifoMode (USART_HandleTypeDef * husart)

Function description

Disable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetTxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - USART_TXFIFO_THRESHOLD_1_8
  - USART_TXFIFO_THRESHOLD_1_4
  - USART_TXFIFO_THRESHOLD_1_2
  - USART_TXFIFO_THRESHOLD_3_4
  - USART_TXFIFO_THRESHOLD_7_8
  - USART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_USARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetRxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - USART_RXFIFO_THRESHOLD_1_8
  - USART_RXFIFO_THRESHOLD_1_4
  - USART_RXFIFO_THRESHOLD_1_2
  - USART_RXFIFO_THRESHOLD_3_4
  - USART_RXFIFO_THRESHOLD_7_8
  - USART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

63.2 USARTEx Firmware driver defines

The following section lists the various define and macros of the module.

63.2.1 USARTEx

USARTEx

USARTEx FIFO mode

USART_FIFOMODE_DISABLE

FIFO mode disable

USART_FIFOMODE_ENABLE

FIFO mode enable

USARTEx RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

USARTEx Synchronous Slave mode enable

USART_SLAVEMODE_DISABLE

USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE

USART SPI Slave Mode Disable

USARTEx Slave Select Management

USART_NSS_HARD

SPI slave selection depends on NSS input pin

USART_NSS_SOFT

SPI slave is always selected and NSS input pin is ignored

USARTEx TXFIFO threshold level

USART_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

USARTEx Word Length

USART_WORDLENGTH_7B

7-bit long USART frame

USART_WORDLENGTH_8B

8-bit long USART frame

USART_WORDLENGTH_9B

9-bit long USART frame

64 HAL WWDG Generic Driver

64.1 WWDG Firmware driver registers structures

64.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32g4xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window***
  Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::Counter***
  Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

64.1.2 WWDG_HandleTypeDef

WWDG_HandleTypeDef is defined in the stm32g4xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDef::Init***
  WWDG required parameters

64.2 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

64.2.1 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the WWDG_InitTypeDef of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- ***HAL_WWDG_Init***
- ***HAL_WWDG_MspInit***

64.2.2 IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- ***HAL_WWDG_Refresh***
- ***HAL_WWDG_IRQHandler***
- ***HAL_WWDG_EarlyWakeupCallback***

64.2.3 Detailed description of functions

HAL_WWDG_Init

Function name

HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG according to the specified.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_MspInit

Function name

void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG MSP.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_Refresh

Function name

HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)

Function description

Refresh the WWDG.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_IRQHandler

Function name

void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)

Function description

Handle WWDG interrupt request.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name

void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)

Function description

WWDG Early Wakeup callback.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

64.3 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

64.3.1 WWDG

WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE

EWI Disable

WWDG_EWI_ENABLE

EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

- Check whether the specified WWDG flag is set or not.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_FLAG

Description:

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_SOURCE

Description:

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF

Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI

Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

65 LL ADC Generic Driver

65.1 ADC Firmware driver registers structures

65.1.1 LL_ADC_CommonInitTypeDef

LL_ADC_CommonInitTypeDef is defined in the stm32g4xx_ll_adc.h

Data Fields

- ***uint32_t CommonClock***
- ***uint32_t Multimode***
- ***uint32_t MultiDMATransfer***
- ***uint32_t MultiTwoSamplingDelay***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::CommonClock***

  Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_COMMON_CLOCK_SOURCE***

  Note:

  – On this STM32 serie, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual.

  This feature can be modified afterwards using unitary function **LL_ADC_SetCommonClock()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::Multimode***

  Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances). This parameter can be a value of ***ADC_LL_EC_MULTI_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetMultimode()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiDMATransfer***

  Set ADC multimode conversion data transfer: no transfer or transfer by DMA. This parameter can be a value of ***ADC_LL_EC_MULTI_DMA_TRANSFER***This feature can be modified afterwards using unitary function **LL_ADC_SetMultiDMATransfer()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiTwoSamplingDelay***

  Set ADC multimode delay between 2 sampling phases. This parameter can be a value of ***ADC_LL_EC_MULTI_TWOSMP_DELAY***This feature can be modified afterwards using unitary function **LL_ADC_SetMultiTwoSamplingDelay()**.

65.1.2 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32g4xx_ll_adc.h

Data Fields

- ***uint32_t Resolution***
- ***uint32_t DataAlignment***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Resolution***

  Set ADC resolution. This parameter can be a value of ***ADC_LL_EC_RESOLUTION***This feature can be modified afterwards using unitary function **LL_ADC_SetResolution()**.
- ***uint32_t LL_ADC_InitTypeDef::DataAlignment***

  Set ADC conversion data alignment. This parameter can be a value of ***ADC_LL_EC_DATA_ALIGN***This feature can be modified afterwards using unitary function **LL_ADC_SetDataAlignment()**.
- ***uint32_t LL_ADC_InitTypeDef::LowPowerMode***

  Set ADC low power mode. This parameter can be a value of ***ADC_LL_EC_LP_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetLowPowerMode()**.

65.1.3 LL_ADC_REG_InitTypeDef

LL_ADC_REG_InitTypeDef is defined in the stm32g4xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DMATransfer***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***

  Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***

  Note:

  – On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_REG_SetTriggerEdge()**.

  This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***

  Set ADC group regular sequencer length. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_SCAN_LENGTH***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***

  Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***

  Note:

  – This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more).

  This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***

  Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode.This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::DMATransfer***

  Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode. This parameter can be a value of ***ADC_LL_EC_REG_DMA_TRANSFER***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDMATransfer()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::Overrun***

  Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of ***ADC_LL_EC_REG_OVR_DATA_BEHAVIOR***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetOverrun()**.

65.1.4 LL_ADC_INJ_InitTypeDef

LL_ADC_INJ_InitTypeDef is defined in the stm32g4xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***

- ***uint32_t TrigAuto***

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***

  Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_INJ_TRIGGER_SOURCE***

  Note:

  – On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_INJ_SetTriggerEdge()**.

  This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***

  Set ADC group injected sequencer length. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_SCAN_LENGTH***This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***

  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_DISCONT_MODE***

  Note:

  – This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more).

  This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***

  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of ***ADC_LL_EC_INJ_TRIG_AUTO*** Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger.This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

65.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

65.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (ADC_TypeDef * ADCx, uint32_t Register)

Function description

Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **ADCx:** ADC instance
- **Register:** This parameter can be one of the following values:
  – LL_ADC_DMA_REG_REGULAR_DATA
  – LL_ADC_DMA_REG_REGULAR_DATA_MULTI (1)

  (1) Available on devices with several ADC instances.

Return values

- **ADC:** register address

Notes

- These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
- For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_DMA_GetRegAddr
- CDR RDATA_MST LL_ADC_DMA_GetRegAddr
- CDR RDATA_SLV LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonClock

Function name

__STATIC_INLINE void LL_ADC_SetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)

Function description

Set parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **CommonClock:** This parameter can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Return values

- **None:**

Notes

- On this STM32 serie, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual.
- On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_SetCommonClock
- CCR PRESC LL_ADC_SetCommonClock

LL_ADC_GetCommonClock

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_GetCommonClock
- CCR PRESC LL_ADC_GetCommonClock

LL_ADC_SetCommonPathInternalCh

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR) The values not selected are removed from configuration.
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalCh
- CCR VSENSESEL LL_ADC_SetCommonPathInternalCh
- CCR VBATSEL LL_ADC_SetCommonPathInternalCh

LL_ADC_SetCommonPathInternalChAdd

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChAdd (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChAdd
- CCR VSENSESEL LL_ADC_SetCommonPathInternalChAdd
- CCR VBATSEL LL_ADC_SetCommonPathInternalChAdd

LL_ADC_SetCommonPathInternalChRem

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChRem (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChRem
- CCR VSENSESEL LL_ADC_SetCommonPathInternalChRem
- CCR VBATSEL LL_ADC_SetCommonPathInternalChRem

LL_ADC_GetCommonPathInternalCh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_GetCommonPathInternalCh
- CCR VSENSESEL LL_ADC_GetCommonPathInternalCh
- CCR VBATSEL LL_ADC_GetCommonPathInternalCh

LL_ADC_SetCalibrationFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED
  - LL_ADC_BOTH_SINGLE_DIFF_ENDED
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F

Return values

- **None:**

Notes

- This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration factor must be specified for each of these differential modes, if used afterwards and if the application requires their calibration).
- In case of setting calibration factors of both modes single ended and differential (parameter LL_ADC_BOTH_SINGLE_DIFF_ENDED): both calibration factors must be concatenated. To perform this processing, use helper macro __LL_ADC_CALIB_FACTOR_SINGLE_DIFF().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_SetCalibrationFactor
- CALFACT CALFACT_D LL_ADC_SetCalibrationFactor

LL_ADC_GetCalibrationFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7F

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_GetCalibrationFactor
- CALFACT CALFACT_D LL_ADC_GetCalibrationFactor

LL_ADC_SetResolution

Function name

__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)

Function description

Set ADC resolution.

Parameters

- **ADCx:** ADC instance
- **Resolution:** This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name

__STATIC_INLINE uint32_t LL_ADC_GetResolution (ADC_TypeDef * ADCx)

Function description

Get ADC resolution.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_GetResolution

LL_ADC_SetDataAlignment

Function name

__STATIC_INLINE void LL_ADC_SetDataAlignment (ADC_TypeDef * ADCx, uint32_t DataAlignment)

Function description

Set ADC conversion data alignment.

Parameters

- **ADCx:** ADC instance
- **DataAlignment:** This parameter can be one of the following values:
  - LL_ADC_DATA_ALIGN_RIGHT
  - LL_ADC_DATA_ALIGN_LEFT

Return values

- **None:**

Notes

- Refer to reference manual for alignments formats dependencies to ADC resolutions.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name

__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (ADC_TypeDef * ADCx)

Function description

Get ADC conversion data alignment.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_DATA_ALIGN_RIGHT
  - LL_ADC_DATA_ALIGN_LEFT

Notes

- Refer to reference manual for alignments formats dependencies to ADC resolutions.

Reference Manual to LL API cross reference:

- CFGR ALIGN LL_ADC_GetDataAlignment

LL_ADC_SetLowPowerMode

Function name

__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Return values

- **None:**

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer.Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_SetLowPowerMode

LL_ADC_GetLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode (ADC_TypeDef * ADCx)

Function description

Get ADC low power mode:

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: Do not use with interruption or DMA since these modes have to clear immediately the EOC flag to free the IRQ vector sequencer.Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_GetLowPowerMode

LL_ADC_SetOffset

Function name

__STATIC_INLINE void LL_ADC_SetOffset (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t Channel, uint32_t OffsetLevel)

Function description

Set ADC selected offset number 1, 2, 3 or 4.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.

- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  –
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- This function set the 2 items of offset configuration: ADC channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)Offset level (offset to be subtracted from the raw converted data).
- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.
- This function enables the offset, by default. It can be forced to disable state using function LL_ADC_SetOffsetState().
- If a channel is mapped on several offsets numbers, only the offset with the lowest value is considered for the subtraction.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On STM32G4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_SetOffset
- OFR1 OFFSET1 LL_ADC_SetOffset
- OFR1 OFFSET1_EN LL_ADC_SetOffset
- OFR2 OFFSET2_CH LL_ADC_SetOffset
- OFR2 OFFSET2 LL_ADC_SetOffset
- OFR2 OFFSET2_EN LL_ADC_SetOffset
- OFR3 OFFSET3_CH LL_ADC_SetOffset
- OFR3 OFFSET3 LL_ADC_SetOffset
- OFR3 OFFSET3_EN LL_ADC_SetOffset
- OFR4 OFFSET4_CH LL_ADC_SetOffset
- OFR4 OFFSET4 LL_ADC_SetOffset
- OFR4 OFFSET4_EN LL_ADC_SetOffset

LL_ADC_GetOffsetChannel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetChannel (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - – LL_ADC_OFFSET_1
  - – LL_ADC_OFFSET_2
  - – LL_ADC_OFFSET_3
  - – LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  - (1, 2, 3, 4, 5, 7) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().
- On STM32G4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_GetOffsetChannel
- OFR2 OFFSET2_CH LL_ADC_GetOffsetChannel
- OFR3 OFFSET3_CH LL_ADC_GetOffsetChannel
- OFR4 OFFSET4_CH LL_ADC_GetOffsetChannel

LL_ADC_GetOffsetLevel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetLevel (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1 LL_ADC_GetOffsetLevel
- OFR2 OFFSET2 LL_ADC_GetOffsetLevel
- OFR3 OFFSET3 LL_ADC_GetOffsetLevel
- OFR4 OFFSET4 LL_ADC_GetOffsetLevel

LL_ADC_SetOffsetState

Function name

__STATIC_INLINE void LL_ADC_SetOffsetState (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetState)

Function description

Set for the ADC selected offset number 1, 2, 3 or 4: force offset state disable or enable without modifying offset channel or offset value.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetState:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_DISABLE
  - LL_ADC_OFFSET_ENABLE

Return values

- **None:**

Notes

- This function should be needed only in case of offset to be enabled-disabled dynamically, and should not be needed in other cases: function LL_ADC_SetOffset() automatically enables the offset.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_EN LL_ADC_SetOffsetState
- OFR2 OFFSET2_EN LL_ADC_SetOffsetState
- OFR3 OFFSET3_EN LL_ADC_SetOffsetState
- OFR4 OFFSET4_EN LL_ADC_SetOffsetState

LL_ADC_GetOffsetState

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetState (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: offset state disabled or enabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_DISABLE
  - LL_ADC_OFFSET_ENABLE

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_EN LL_ADC_GetOffsetState
- OFR2 OFFSET2_EN LL_ADC_GetOffsetState
- OFR3 OFFSET3_EN LL_ADC_GetOffsetState
- OFR4 OFFSET4_EN LL_ADC_GetOffsetState

LL_ADC_SetOffsetSign

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSign (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSign)

Function description

Set for the ADC selected offset number 1, 2, 3 or 4: choose offset sign.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSign:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSETPOS LL_ADC_SetOffsetSign
- OFR2 OFFSETPOS LL_ADC_SetOffsetSign
- OFR3 OFFSETPOS LL_ADC_SetOffsetSign
- OFR4 OFFSETPOS LL_ADC_SetOffsetSign

LL_ADC_GetOffsetSign

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSign (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: offset sign if positive or negative.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Reference Manual to LL API cross reference:

- OFR1 OFFSETPOS LL_ADC_GetOffsetSign
- OFR2 OFFSETPOS LL_ADC_GetOffsetSign
- OFR3 OFFSETPOS LL_ADC_GetOffsetSign
- OFR4 OFFSETPOS LL_ADC_GetOffsetSign

LL_ADC_SetOffsetSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSaturation)

Function description

Set for the ADC selected offset number 1, 2, 3 or 4: choose offset saturation mode.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSaturation:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SATURATION_ENABLE
  - LL_ADC_OFFSET_SATURATION_DISABLE

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 SATEN LL_ADC_SetOffsetSaturation
- OFR2 SATEN LL_ADC_SetOffsetSaturation
- OFR3 SATEN LL_ADC_SetOffsetSaturation
- OFR4 SATEN LL_ADC_SetOffsetSaturation

LL_ADC_GetOffsetSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSaturation (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SATURATION_ENABLE
  - LL_ADC_OFFSET_SATURATION_DISABLE

Reference Manual to LL API cross reference:

- OFR1 SATEN LL_ADC_GetOffsetSaturation
- OFR2 SATEN LL_ADC_GetOffsetSaturation
- OFR3 SATEN LL_ADC_GetOffsetSaturation
- OFR4 SATEN LL_ADC_GetOffsetSaturation

LL_ADC_SetGainCompensation

Function name

__STATIC_INLINE void LL_ADC_SetGainCompensation (ADC_TypeDef * ADCx, uint32_t GainCompensation)

Function description

Set ADC gain compensation.

Parameters

- **ADCx:** ADC instance
- **GainCompensation:** This parameter can be: 0 Gain compensation will be disabled and value set to 0 1 -> 16393 Gain compensation will be enabled with specified value

Return values

- **None:**

Notes

- This function set the gain compensation coefficient that is applied to raw converted data using the formula: DATA = DATA(raw) * (gain compensation coef) / 4096
- This function enables the gain compensation if given coefficient is above 0, otherwise it disables it.
- Gain compensation when enabled is appied to all channels.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- GCOMP GCOMPCOEFF LL_ADC_SetGainCompensation
- CFGR2 GCOMP LL_ADC_SetGainCompensation

LL_ADC_GetGainCompensation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetGainCompensation (ADC_TypeDef * ADCx)

Function description

Get the ADC gain compensation value.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be: 0 Gain compensation is disabled 1 -> 16393 Gain compensation is enabled with returned value

Reference Manual to LL API cross reference:

- GCOMP GCOMPCOEFF LL_ADC_GetGainCompensation
- CFGR2 GCOMP LL_ADC_GetGainCompensation

LL_ADC_SetSamplingTimeCommonConfig

Function name

__STATIC_INLINE void LL_ADC_SetSamplingTimeCommonConfig (ADC_TypeDef * ADCx, uint32_t SamplingTimeCommonConfig)

Function description

Set ADC sampling time common configuration impacting settings of sampling time channel wise.

Parameters

- **ADCx:** ADC instance
- **SamplingTimeCommonConfig:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_DEFAULT
  - LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMPPLUS LL_ADC_SetSamplingTimeCommonConfig

LL_ADC_GetSamplingTimeCommonConfig

Function name

__STATIC_INLINE uint32_t LL_ADC_GetSamplingTimeCommonConfig (ADC_TypeDef * ADCx)

Function description

Get ADC sampling time common configuration impacting settings of sampling time channel wise.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_DEFAULT
  - LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5

Reference Manual to LL API cross reference:

- SMPR1 SMPPLUS LL_ADC_GetSamplingTimeCommonConfig

LL_ADC_REG_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1 (1)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM2_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH3 (2)
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4 (1)
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4 (1)
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM7_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM8_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM20_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM20_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM20_CH1
  - LL_ADC_REG_TRIG_EXT_TIM20_CH2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM20_CH3 (1)
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG1
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG2 (2)
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG3
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG4 (2)
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG5
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG6
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG7
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG8
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG9
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG10
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11 (1)
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE2 (2)
  - LL_ADC_REG_TRIG_EXT_LPTIM_OUT

  (1) On STM32G4 serie, parameter not available on all ADC instances: ADC1, ADC2.
- (2) On STM32G4 serie, parameter not available on all ADC instances: ADC3, ADC4, ADC5. On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Return values

- **None:**

Notes

- On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_SetTriggerSource
- CFGR EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1 (1)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM2_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM2_CH3 (2)
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4 (1)
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4 (1)
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM7_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM8_CH1 (2)
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM20_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM20_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM20_CH1
  - LL_ADC_REG_TRIG_EXT_TIM20_CH2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM20_CH3 (1)
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG1
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG2 (2)
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG3
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG4 (2)
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG5
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG6
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG7
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG8
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG9
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG10
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11 (1)
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE2 (2)
  - LL_ADC_REG_TRIG_EXT_LPTIM_OUT

  (1) On STM32G4 serie, parameter not available on all ADC instances: ADC1, ADC2.
- (2) On STM32G4 serie, parameter not available on all ADC instances: ADC3, ADC4, ADC5. On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Notes

- To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_GetTriggerSource
- CFGR EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- Applicable only for trigger source set to external trigger.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Notes

- Applicable only for trigger source set to external trigger.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_REG_SetSamplingMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetSamplingMode (ADC_TypeDef * ADCx, uint32_t SamplingMode)

Function description

Set ADC sampling mode.

Parameters

- **ADCx:** ADC instance
- **SamplingMode:** This parameter can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Return values

- **None:**

Notes

- This function set the ADC conversion sampling mode
- This mode applies to regular group only.
- Set sampling mode is appied to all conversion of regular group.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_SetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_SetSamplingMode

LL_ADC_REG_GetSamplingMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSamplingMode (ADC_TypeDef * ADCx)

Function description

Get the ADC sampling mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_GetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_GetSamplingMode

LL_ADC_REG_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Return values

- **None:**

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_GetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Return values

- **None:**

Notes

- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_SetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_SetSequencerDiscont

LL_ADC_REG_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_GetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16

- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  –

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 serie, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  - (1, 2, 3, 4, 5, 7) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- On this STM32 serie, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_REG_SetContinuousMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)

Function description

Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_SetContinuousMode

LL_ADC_REG_GetContinuousMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (ADC_TypeDef * ADCx)

Function description

Get ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDMATransfer

Function name

__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)

Function description

Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters

- **ADCx:** ADC instance
- **DMATransfer:** This parameter can be one of the following values:
  - LL_ADC_REG_DMA_TRANSFER_NONE
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Return values

- **None:**

Notes

- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- For devices with several ADC instances: ADC multimode DMA settings are available using function LL_ADC_SetMultiDMATransfer().
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR DMAEN LL_ADC_REG_SetDMATransfer
- CFGR DMACFG LL_ADC_REG_SetDMATransfer

LL_ADC_REG_GetDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_DMA_TRANSFER_NONE
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Notes

- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- For devices with several ADC instances: ADC multimode DMA settings are available using function LL_ADC_GetMultiDMATransfer().
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CFGR DMAEN LL_ADC_REG_GetDMATransfer
- CFGR DMACFG LL_ADC_REG_GetDMATransfer

LL_ADC_REG_SetOverrun

Function name

__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)

Function description

Set ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance
- **Overrun:** This parameter can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Return values

- **None:**

Notes

- Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (ADC_TypeDef * ADCx)

Function description

Get ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_INJ_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH3 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH3 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH4 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH2 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM16_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM20_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM20_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM20_CH2 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM20_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG1 (2)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG3 (2)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG5
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG6
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG7
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG8
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG9
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG10
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE3 (2)
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15 (1)
  - LL_ADC_INJ_TRIG_EXT_LPTIM_OUT

  (1) On STM32G4 serie, parameter not available on all ADC instances: ADC1, ADC2.
- (2) On STM32G4 serie, parameter not available on all ADC instances: ADC3, ADC4, ADC5. On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Return values

- **None:**

Notes

- On this STM32 serie, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_INJ_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_SetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH3 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH3 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH4 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH2 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM16_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM20_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM20_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM20_CH2 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM20_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG1 (2)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG3 (2)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG5
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG6
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG7
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG8
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG9
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG10
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE3 (2)
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15 (1)
  - LL_ADC_INJ_TRIG_EXT_LPTIM_OUT

  (1) On STM32G4 serie, parameter not available on all ADC instances: ADC1, ADC2.
- (2) On STM32G4 serie, parameter not available on all ADC instances: ADC3, ADC4, ADC5. On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Notes

- To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_GetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_SetTriggerEdge

LL_ADC_INJ_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerEdge (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_GetTriggerEdge

LL_ADC_INJ_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Return values

- **None:**

Notes

- This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Notes

- This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Return values

- **None:**

Notes

- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.

- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
- –

Return values

- **None:**

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32G4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  - (1, 2, 3, 4, 5, 7) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_GetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description

Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values

- **None:**

Notes

- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.
- If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_INJ_SetQueueMode

Function name

__STATIC_INLINE void LL_ADC_INJ_SetQueueMode (ADC_TypeDef * ADCx, uint32_t QueueMode)

Function description

Set ADC group injected contexts queue mode.

Parameters

- **ADCx:** ADC instance
- **QueueMode:** This parameter can be one of the following values:
  - LL_ADC_INJ_QUEUE_DISABLE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks If contexts queue is disabled:only 1 sequence can be configured and is active perpetually. If contexts queue is enabled:up to 2 contexts can be queued and are checked in and out as a FIFO stack (first-in, first-out).If a new context is set when queues is full, error is triggered by interruption "Injected Queue Overflow".Two behaviors are possible when all contexts have been processed: the contexts queue can maintain the last context active perpetually or can be empty and injected group triggers are disabled.Triggers can be only external (not internal SW start)Caution: The sequence must be fully configured in one time (one write of register JSQR makes a check-in of a new context into the queue). Therefore functions to set separately injected trigger and sequencer channels cannot be used, register JSQR must be set using function LL_ADC_INJ_ConfigQueueContext().
- This parameter can be modified only when no conversion is on going on either groups regular or injected.
- A modification of the context mode (bit JQDIS) causes the contexts queue to be flushed and the register JSQR is cleared.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JQM LL_ADC_INJ_SetQueueMode
- CFGR JQDIS LL_ADC_INJ_SetQueueMode

LL_ADC_INJ_GetQueueMode

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetQueueMode (ADC_TypeDef * ADCx)

Function description

Get ADC group injected context queue mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_QUEUE_DISABLE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Reference Manual to LL API cross reference:

- CFGR JQM LL_ADC_INJ_GetQueueMode
- CFGR JQDIS LL_ADC_INJ_GetQueueMode

LL_ADC_INJ_ConfigQueueContext

Function name

__STATIC_INLINE void LL_ADC_INJ_ConfigQueueContext (ADC_TypeDef * ADCx, uint32_t TriggerSource, uint32_t ExternalTriggerEdge, uint32_t SequencerNbRanks, uint32_t Rank1_Channel, uint32_t Rank2_Channel, uint32_t Rank3_Channel, uint32_t Rank4_Channel)

Function description

Set one context on ADC group injected that will be checked in contexts queue.

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH3 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH3 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM4_CH4 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH2 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM16_CH1 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM20_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM20_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM20_CH2 (2)
  - LL_ADC_INJ_TRIG_EXT_TIM20_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG1 (2)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG3 (2)
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG5
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG6
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG7
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG8
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG9
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG10
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE3 (2)
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15 (1)
  - LL_ADC_INJ_TRIG_EXT_LPTIM_OUT

  (1) On STM32G4 serie, parameter not available on all ADC instances: ADC1, ADC2.
- (2) On STM32G4 serie, parameter not available on all ADC instances: ADC3, ADC4, ADC5. On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

  Note: This parameter is discarded in case of SW start: parameter "TriggerSource" set to "LL_ADC_INJ_TRIG_SOFTWARE".
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS
- **Rank1_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.

- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  –
- **Rank2_Channel:** This parameter can be one of the following values:
  – LL_ADC_CHANNEL_0
  – LL_ADC_CHANNEL_1 (8)
  – LL_ADC_CHANNEL_2 (8)
  – LL_ADC_CHANNEL_3 (8)
  – LL_ADC_CHANNEL_4 (8)
  – LL_ADC_CHANNEL_5 (8)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_VREFINT (7)
  – LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  – LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  – LL_ADC_CHANNEL_VBAT (6)
  – LL_ADC_CHANNEL_VOPAMP1 (1)
  – LL_ADC_CHANNEL_VOPAMP2 (2)
  – LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  – LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  – LL_ADC_CHANNEL_VOPAMP4 (5)
  – LL_ADC_CHANNEL_VOPAMP5 (5)
  – LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  –

- **Rank3_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
- –

- **Rank4_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
- –

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks This function is intended to be used when contexts queue is enabled, because the sequence must be fully configured in one time (functions to set separately injected trigger and sequencer channels cannot be used): Refer to function LL_ADC_INJ_SetQueueMode().
- In the contexts queue, only the active context can be read. The parameters of this function can be read using functions: LL_ADC_INJ_GetTriggerSource() LL_ADC_INJ_GetTriggerEdge() LL_ADC_INJ_GetSequencerRanks()
- On this STM32 serie, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32G4, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN1..5).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_ConfigQueueContext
- JSQR JEXTEN LL_ADC_INJ_ConfigQueueContext
- JSQR JL LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ1 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ2 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ3 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ4 LL_ADC_INJ_ConfigQueueContext

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  –

- **SamplingTime:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_2CYCLES_5 (1)
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_24CYCLES_5
  - LL_ADC_SAMPLINGTIME_47CYCLES_5
  - LL_ADC_SAMPLINGTIME_92CYCLES_5
  - LL_ADC_SAMPLINGTIME_247CYCLES_5
  - LL_ADC_SAMPLINGTIME_640CYCLES_5

  (1) On some devices, ADC sampling time 2.5 ADC clock cycles can be replaced by 3.5 ADC clock cycles. Refer to function LL_ADC_SetSamplingTimeCommonConfig().

Return values

- **None:**

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 serie, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)

  (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.
  –

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_2CYCLES_5 (1)
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_24CYCLES_5
  - LL_ADC_SAMPLINGTIME_47CYCLES_5
  - LL_ADC_SAMPLINGTIME_92CYCLES_5
  - LL_ADC_SAMPLINGTIME_247CYCLES_5
  - LL_ADC_SAMPLINGTIME_640CYCLES_5

  (1) On some devices, ADC sampling time 2.5 ADC clock cycles can be replaced by 3.5 ADC clock cycles. Refer to function LL_ADC_SetSamplingTimeCommonConfig().

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. On this STM32 serie, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_GetChannelSamplingTime

LL_ADC_SetChannelSingleDiff

Function name

__STATIC_INLINE void LL_ADC_SetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SingleDiff)

Function description

Set mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
- **SingleDiff:** This parameter can be a combination of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically.
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32G4, some channels are internally fixed to single-ended inputs configuration: ADC1: Channels 12, 15, 16, 17 and 18ADC2: Channels 15, 17 and 18ADC3: Channels 12, 16, 17 and 18 (1)ADC4: Channels 16, 17 and 18 (1)ADC5: Channels 2, 3, 4, 16, 17 and 18 (1) (1) ADC3/4/5 are not available on all devices, refer to device datasheet for more details.
- For ADC channels configured in differential mode, both inputs should be biased at (Vref+)/2 +/-200mV. (Vref+ is the analog voltage reference)
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_SetChannelSingleDiff

LL_ADC_GetChannelSingleDiff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15

Return values

- **0:** channel in single-ended mode, else: channel in differential mode

Notes

- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Therefore, to ensure a channel is configured in single-ended mode, the configuration of channel itself and the channel 'i-1' must be read back (to ensure that the selected channel channel has not been configured in differential mode by the previous channel).
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32G4, some channels are internally fixed to single-ended inputs configuration: ADC1: Channels 12, 15, 16, 17 and 18ADC2: Channels 15, 17 and 18ADC3: Channels 12, 16, 17 and 18 (1)ADC4: Channels 16, 17 and 18 (1)ADC5: Channels 2, 3, 4, 16, 17 and 18 (1) (1) ADC3/4/5 are not available on all devices, refer to device datasheet for more details.
- One or several values can be selected. In this case, the value returned is null if all channels are in single ended-mode. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_GetChannelSingleDiff

LL_ADC_SetAnalogWDMonitChannels

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDChannelGroup)

Function description

Set ADC analog watchdog monitored channels: a single channel, multiple channels or all channels, on ADC groups regular and-or injected.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3

- **AWDChannelGroup:** This parameter can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ
  - LL_ADC_AWD_CHANNEL_15_REG (0)
  - LL_ADC_AWD_CHANNEL_15_INJ (0)

- (1) On STM32G4, parameter available only on ADC instance: ADC1.
- (2) On STM32G4, parameter available only on ADC instance: ADC2.
- (3) On STM32G4, parameter available only on ADC instance: ADC3.
- (4) On STM32G4, parameter available only on ADC instance: ADC4.
- (5) On STM32G4, parameter available only on ADC instance: ADC5.
- (6) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC5.
- (7) On STM32G4, parameter available only on ADC instances: ADC1, ADC3, ADC4, ADC5.
- – On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Return values

- **None:**

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_SetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_SetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_SetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog monitored channel.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2 (1)
  - LL_ADC_AWD3 (1)

  (1) On this AWD number, monitored channel can be retrieved if only 1 channel is programmed (or none or all channels). This function cannot retrieve monitored channel if multiple channels are programmed simultaneously by bitfield.

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ
  - LL_ADC_AWD_CHANNEL_15_REG (0)

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_GetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_GetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

LL_ADC_ConfigAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_ConfigAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdHighValue, uint32_t AWDThresholdLowValue)

Function description

Set ADC analog watchdog thresholds value of both thresholds high and low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdHighValue:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **AWDThresholdLowValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- If value of only one threshold high or low must be set, use function LL_ADC_SetAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 HT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 HT3 LL_ADC_ConfigAnalogWDThresholds
- TR1 LT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 LT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 LT3 LL_ADC_ConfigAnalogWDThresholds

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 serie, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).
- On this STM32 serie, setting of this feature is not conditioned to ADC state: ADC can be disabled, enabled with or without conversion on going on either ADC groups regular or injected.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_SetAnalogWDThresholds
- TR2 HT2 LL_ADC_SetAnalogWDThresholds
- TR3 HT3 LL_ADC_SetAnalogWDThresholds
- TR1 LT1 LL_ADC_SetAnalogWDThresholds
- TR2 LT2 LL_ADC_SetAnalogWDThresholds
- TR3 LT3 LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow)

Function description

Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
  - LL_ADC_AWD_THRESHOLDS_HIGH_LOW

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_GetAnalogWDThresholds
- TR2 HT2 LL_ADC_GetAnalogWDThresholds
- TR3 HT3 LL_ADC_GetAnalogWDThresholds
- TR1 LT1 LL_ADC_GetAnalogWDThresholds
- TR2 LT2 LL_ADC_GetAnalogWDThresholds
- TR3 LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_SetAWDFilteringConfiguration

Function name

__STATIC_INLINE void LL_ADC_SetAWDFilteringConfiguration (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t FilteringConfig)

Function description

Set ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
- **FilteringConfig:** This parameter can be one of the following values:
  - LL_ADC_AWD_FILTERING_NONE
  - LL_ADC_AWD_FILTERING_2SAMPLES
  - LL_ADC_AWD_FILTERING_3SAMPLES
  - LL_ADC_AWD_FILTERING_4SAMPLES
  - LL_ADC_AWD_FILTERING_5SAMPLES
  - LL_ADC_AWD_FILTERING_6SAMPLES
  - LL_ADC_AWD_FILTERING_7SAMPLES
  - LL_ADC_AWD_FILTERING_8SAMPLES

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On this STM32 serie, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- TR1 AWDFILT LL_ADC_SetAWDFilteringConfiguration

LL_ADC_GetAWDFilteringConfiguration

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDFilteringConfiguration (ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1

Return values

- **Returned:** value can be:
  - LL_ADC_AWD_FILTERING_NONE
  - LL_ADC_AWD_FILTERING_2SAMPLES
  - LL_ADC_AWD_FILTERING_3SAMPLES
  - LL_ADC_AWD_FILTERING_4SAMPLES
  - LL_ADC_AWD_FILTERING_5SAMPLES
  - LL_ADC_AWD_FILTERING_6SAMPLES
  - LL_ADC_AWD_FILTERING_7SAMPLES
  - LL_ADC_AWD_FILTERING_8SAMPLES

Notes

- On this STM32 serie, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- TR1 AWDFILT LL_ADC_GetAWDFilteringConfiguration

LL_ADC_SetOverSamplingScope

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)

Function description

Set ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).

Parameters

- **ADCx:** ADC instance
- **OvsScope:** This parameter can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Return values

- **None:**

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_SetOverSamplingScope
- CFGR2 JOVSE LL_ADC_SetOverSamplingScope
- CFGR2 ROVSM LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_GetOverSamplingScope
- CFGR2 JOVSE LL_ADC_GetOverSamplingScope
- CFGR2 ROVSM LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)

Function description

Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance
- **OverSamplingDiscont:** This parameter can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Return values

- **None:**

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- On this STM32 serie, oversampling discontinuous mode (triggered mode) can be used only when oversampling is set on group regular only and in resumed mode.

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name

__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)

Function description

Set ADC oversampling (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance
- **Ratio:** This parameter can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256
- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8

Return values

- **None:**

Notes

- This function set the 2 items of oversampling configuration: ratioshift
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift
- CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling ratio (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Ratio:** This parameter can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256

Reference Manual to LL API cross reference:

- CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling shift (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_SetMultimode

Function name

__STATIC_INLINE void LL_ADC_SetMultimode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Multimode)

Function description

Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **Multimode:** This parameter can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Return values

- **None:**

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.
- On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_SetMultimode

LL_ADC_GetMultimode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultimode (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_GetMultimode

LL_ADC_SetMultiDMATransfer

Function name

__STATIC_INLINE void LL_ADC_SetMultiDMATransfer (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiDMATransfer)

Function description

Set ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiDMATransfer:** This parameter can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES8_6B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES8_6B

Return values

- **None:**

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().
- On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CCR MDMA LL_ADC_SetMultiDMATransfer
- CCR DMACFG LL_ADC_SetMultiDMATransfer

LL_ADC_GetMultiDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiDMATransfer (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES8_6B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES8_6B

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().

Reference Manual to LL API cross reference:

- CCR MDMA LL_ADC_GetMultiDMATransfer
- CCR DMACFG LL_ADC_GetMultiDMATransfer

LL_ADC_SetMultiTwoSamplingDelay

Function name

__STATIC_INLINE void LL_ADC_SetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiTwoSamplingDelay)

Function description

Set ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiTwoSamplingDelay:** This parameter can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES (3)
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES (3)

  (1) Parameter available only if ADC resolution is 12, 10 or 8 bits.
- (2) Parameter available only if ADC resolution is 12 or 10 bits.
- (3) Parameter available only if ADC resolution is 12 bits.

Return values

- **None:**

Notes

- The sampling delay range depends on ADC resolution: ADC resolution 12 bits can have maximum delay of 12 cycles.ADC resolution 10 bits can have maximum delay of 10 cycles.ADC resolution 8 bits can have maximum delay of 8 cycles.ADC resolution 6 bits can have maximum delay of 6 cycles.
- On this STM32 serie, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_SetMultiTwoSamplingDelay

LL_ADC_GetMultiTwoSamplingDelay

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES (3)
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES (3)

  (1) Parameter available only if ADC resolution is 12, 10 or 8 bits.
- (2) Parameter available only if ADC resolution is 12 or 10 bits.
- (3) Parameter available only if ADC resolution is 12 bits.

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_GetMultiTwoSamplingDelay

LL_ADC_EnableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_EnableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Put ADC instance in deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_EnableDeepPowerDown

LL_ADC_DisableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_DisableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Disable ADC deep power down mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_DisableDeepPowerDown

LL_ADC_IsDeepPowerDownEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDeepPowerDownEnabled (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** deep power down is disabled, 1: deep power down is enabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_IsDeepPowerDownEnabled

LL_ADC_EnableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Enable ADC instance internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter tADCVREG_STUP. Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Disable ADC internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance internal voltage regulator state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** internal regulator is disabled, 1: internal regulator is enabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled

LL_ADC_Enable

Function name

__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description

Enable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.
- On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name

__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)

Function description

Disable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance enable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC is disabled, 1: ADC is enabled.

Notes

- On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_IsEnabled

LL_ADC_IsDisableOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance disable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name

__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- On this STM32 serie, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES.
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration run must be performed for each of these differential modes, if used afterwards and if the application requires their calibration).
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StartCalibration
- CR ADCALDIF LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (ADC_TypeDef * ADCx)

Function description

Get ADC calibration state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration complete, 1: calibration in progress.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group regular command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_StartSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StartSamplingPhase (ADC_TypeDef * ADCx)

Function description

Start ADC sampling phase for sampling time trigger mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger source
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StartSamplingPhase

LL_ADC_REG_StopSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StopSamplingPhase (ADC_TypeDef * ADCx)

Function description

Stop ADC sampling phase for sampling time trigger mode and start conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger sourceLL_ADC_REG_StartSamplingPhase has been called to start the sampling phase
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StopSamplingPhase

LL_ADC_REG_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData6

LL_ADC_REG_ReadMultiConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConversionData32 (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t ConversionData)

Function description

Get ADC multimode conversion data of ADC master, ADC slave or raw data with ADC master and slave concatenated.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ConversionData:** This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
  - LL_ADC_MULTI_MASTER_SLAVE

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- If raw data with ADC master and slave concatenated is retrieved, a macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE(). (however this macro is mainly intended for multimode transfer by DMA, because this function can do the same by getting multimode conversion data of ADC master or ADC slave separately).

Reference Manual to LL API cross reference:

- CDR RDATA_MST LL_ADC_REG_ReadMultiConversionData32
- CDR RDATA_SLV LL_ADC_REG_ReadMultiConversionData32

LL_ADC_INJ_StartConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group injected, without conversion stop command on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_StartConversion

LL_ADC_INJ_StopConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_StopConversion

LL_ADC_INJ_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_IsConversionOngoing

LL_ADC_INJ_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsStopConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group injected command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_IsStopConversionOngoing

LL_ADC_INJ_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData32
- JDR2 JDATA LL_ADC_INJ_ReadConversionData32
- JDR3 JDATA LL_ADC_INJ_ReadConversionData32
- JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData10
- JDR2 JDATA LL_ADC_INJ_ReadConversionData10
- JDR3 JDATA LL_ADC_INJ_ReadConversionData10
- JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData8
- JDR2 JDATA LL_ADC_INJ_ReadConversionData8
- JDR3 JDATA LL_ADC_INJ_ReadConversionData8
- JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_INJ_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData6 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData6
- JDR2 JDATA LL_ADC_INJ_ReadConversionData6
- JDR3 JDATA LL_ADC_INJ_ReadConversionData6
- JDR4 JDATA LL_ADC_INJ_ReadConversionData6

LL_ADC_IsActiveFlag_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Get flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Notes

- On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_IsActiveFlag_JEOC

LL_ADC_IsActiveFlag_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JQOVF (ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected contexts queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JQOVF LL_ADC_IsActiveFlag_JQOVF

LL_ADC_IsActiveFlag_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 1 flag.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_IsActiveFlag_AWD1

LL_ADC_IsActiveFlag_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_IsActiveFlag_AWD2

LL_ADC_IsActiveFlag_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_IsActiveFlag_AWD3

LL_ADC_ClearFlag_ADRDY

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Clear flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 serie, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_JEOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_ClearFlag_JEOC

LL_ADC_ClearFlag_JEOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_ClearFlag_JEOS

LL_ADC_ClearFlag_JQOVF

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JQOVF (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected contexts queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JQOVF LL_ADC_ClearFlag_JQOVF

LL_ADC_ClearFlag_AWD1

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_AWD2

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_ClearFlag_AWD2

LL_ADC_ClearFlag_AWD3

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_ClearFlag_AWD3

LL_ADC_IsActiveFlag_MST_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_ADRDY (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_MST LL_ADC_IsActiveFlag_MST_ADRDY

LL_ADC_IsActiveFlag_SLV_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_ADRDY (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_SLV LL_ADC_IsActiveFlag_SLV_ADRDY

LL_ADC_IsActiveFlag_MST_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_MST LL_ADC_IsActiveFlag_MST_EOC

LL_ADC_IsActiveFlag_SLV_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_SLV LL_ADC_IsActiveFlag_SLV_EOC

LL_ADC_IsActiveFlag_MST_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_MST LL_ADC_IsActiveFlag_MST_EOS

LL_ADC_IsActiveFlag_SLV_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_SLV LL_ADC_IsActiveFlag_SLV_EOS

LL_ADC_IsActiveFlag_MST_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_OVR (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_MST LL_ADC_IsActiveFlag_MST_OVR

LL_ADC_IsActiveFlag_SLV_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_OVR (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_SLV LL_ADC_IsActiveFlag_SLV_OVR

LL_ADC_IsActiveFlag_MST_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOSMP (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_MST LL_ADC_IsActiveFlag_MST_EOSMP

LL_ADC_IsActiveFlag_SLV_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOSMP (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_SLV LL_ADC_IsActiveFlag_SLV_EOSMP

LL_ADC_IsActiveFlag_MST_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_MST LL_ADC_IsActiveFlag_MST_JEOC

LL_ADC_IsActiveFlag_SLV_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_SLV LL_ADC_IsActiveFlag_SLV_JEOC

LL_ADC_IsActiveFlag_MST_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_MST LL_ADC_IsActiveFlag_MST_JEOS

LL_ADC_IsActiveFlag_SLV_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_SLV LL_ADC_IsActiveFlag_SLV_JEOS

LL_ADC_IsActiveFlag_MST_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JQOVF (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected context queue overflow of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JQOVF_MST LL_ADC_IsActiveFlag_MST_JQOVF

LL_ADC_IsActiveFlag_SLV_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JQOVF (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected context queue overflow of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JQOVF_SLV LL_ADC_IsActiveFlag_SLV_JQOVF

LL_ADC_IsActiveFlag_MST_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD1 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 1 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_MST LL_ADC_IsActiveFlag_MST_AWD1

LL_ADC_IsActiveFlag_SLV_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD1 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode analog watchdog 1 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_SLV LL_ADC_IsActiveFlag_SLV_AWD1

LL_ADC_IsActiveFlag_MST_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD2 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_MST LL_ADC_IsActiveFlag_MST_AWD2

LL_ADC_IsActiveFlag_SLV_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD2 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_SLV LL_ADC_IsActiveFlag_SLV_AWD2

LL_ADC_IsActiveFlag_MST_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD3 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_MST LL_ADC_IsActiveFlag_MST_AWD3

LL_ADC_IsActiveFlag_SLV_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD3 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_SLV LL_ADC_IsActiveFlag_SLV_AWD3

LL_ADC_EnableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Enable ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)

Function description

Enable ADC group regular interruption overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_EnableIT_JEOC

LL_ADC_EnableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_JQOVF

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected context queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_EnableIT_JQOVF

LL_ADC_EnableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_EnableIT_AWD2

LL_ADC_EnableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_EnableIT_AWD3

LL_ADC_DisableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_DisableIT_JEOC

LL_ADC_DisableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_DisableIT_JEOS

LL_ADC_DisableIT_JQOVF

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected context queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_DisableIT_JQOVF

LL_ADC_DisableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_DisableIT_AWD2

LL_ADC_DisableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_DisableIT_AWD3

LL_ADC_IsEnabledIT_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOC (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_IsEnabledIT_JEOC

LL_ADC_IsEnabledIT_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_IsEnabledIT_JEOS

LL_ADC_IsEnabledIT_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected context queue overflow interrupt state (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_IsEnabledIT_JQOVF

LL_ADC_IsEnabledIT_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_IsEnabledIT_AWD1

LL_ADC_IsEnabledIT_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 2 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_IsEnabledIT_AWD2

LL_ADC_IsEnabledIT_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 3 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_IsEnabledIT_AWD3

LL_ADC_CommonDeInit

Function name

ErrorStatus LL_ADC_CommonDeInit (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are de-initialized
  - ERROR: not applicable

Notes

- This function is performing a hard reset, using high level clock source RCC ADC reset. Caution: On this STM32 serie, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. To de-initialize only 1 ADC instance, use function LL_ADC_DeInit().

LL_ADC_CommonInit

Function name

ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)

Function description

Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are initialized
  - ERROR: ADC common registers are not initialized

Notes

- The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name

void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)

Function description

Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters

- **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_DeInit

Function name

ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)

Function description

De-initialize registers of the selected ADC instance to their default reset values.

Parameters

- **ADCx:** ADC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are de-initialized
  - ERROR: ADC registers are not de-initialized

Notes

- To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit().
- If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Caution: On this STM32 serie, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name

ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)

Function description

Initialize some features of ADC instance.

Parameters

- **ADCx:** ADC instance
- **ADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 families). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name

void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)

Function description

Set each LL_ADC_InitTypeDef field to default value.

Parameters

- **ADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_REG_Init

Function name

ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)

Function description

Initialize some features of ADC group regular.

Parameters

- **ADCx:** ADC instance
- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name

void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)

Function description

Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters

- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_INJ_Init

Function name

ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * ADCx, LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)

Function description

Initialize some features of ADC group injected.

Parameters

- **ADCx:** ADC instance
- **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_INJ_StructInit

Function name

void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)

Function description

Set each LL_ADC_INJ_InitTypeDef field to default value.

Parameters

- **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

65.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

65.3.1 ADC

ADC

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE

ADC analog watchdog monitoring disabled

LL_ADC_AWD_ALL_CHANNELS_REG

ADC analog watchdog monitoring of all channels, converted by group regular only

LL_ADC_AWD_ALL_CHANNELS_INJ

ADC analog watchdog monitoring of all channels, converted by group injected only

LL_ADC_AWD_ALL_CHANNELS_REG_INJ

ADC analog watchdog monitoring of all channels, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_0_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group regular only

LL_ADC_AWD_CHANNEL_0_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group injected only

LL_ADC_AWD_CHANNEL_0_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_1_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group regular only

LL_ADC_AWD_CHANNEL_1_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group injected only

LL_ADC_AWD_CHANNEL_1_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_2_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group regular only

LL_ADC_AWD_CHANNEL_2_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group injected only

LL_ADC_AWD_CHANNEL_2_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_3_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group regular only

LL_ADC_AWD_CHANNEL_3_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group injected only

LL_ADC_AWD_CHANNEL_3_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_4_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group regular only

LL_ADC_AWD_CHANNEL_4_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group injected only

LL_ADC_AWD_CHANNEL_4_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_5_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group regular only

LL_ADC_AWD_CHANNEL_5_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group injected only

LL_ADC_AWD_CHANNEL_5_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_6_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group regular only

LL_ADC_AWD_CHANNEL_6_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group injected only

LL_ADC_AWD_CHANNEL_6_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_7_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group regular only

LL_ADC_AWD_CHANNEL_7_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group injected only

LL_ADC_AWD_CHANNEL_7_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_8_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group regular only

LL_ADC_AWD_CHANNEL_8_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group injected only

LL_ADC_AWD_CHANNEL_8_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_9_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group regular only

LL_ADC_AWD_CHANNEL_9_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group injected only

LL_ADC_AWD_CHANNEL_9_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_10_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group regular only

LL_ADC_AWD_CHANNEL_10_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group injected only

LL_ADC_AWD_CHANNEL_10_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_11_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group regular only

LL_ADC_AWD_CHANNEL_11_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group injected only

LL_ADC_AWD_CHANNEL_11_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_12_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group regular only

LL_ADC_AWD_CHANNEL_12_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group injected only

LL_ADC_AWD_CHANNEL_12_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_13_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group regular only

LL_ADC_AWD_CHANNEL_13_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group injected only

LL_ADC_AWD_CHANNEL_13_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_14_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group regular only

LL_ADC_AWD_CHANNEL_14_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group injected only

LL_ADC_AWD_CHANNEL_14_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_15_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group regular only

LL_ADC_AWD_CHANNEL_15_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group injected only

LL_ADC_AWD_CHANNEL_15_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_16_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group regular only

LL_ADC_AWD_CHANNEL_16_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group injected only

LL_ADC_AWD_CHANNEL_16_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_17_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only

LL_ADC_AWD_CHANNEL_17_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group injected only

LL_ADC_AWD_CHANNEL_17_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_18_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group regular only

LL_ADC_AWD_CHANNEL_18_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group injected only

LL_ADC_AWD_CHANNEL_18_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by either group regular or injected

LL_ADC_AWD_CH_VREFINT_REG

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only

LL_ADC_AWD_CH_VREFINT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only

LL_ADC_AWD_CH_VREFINT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected

LL_ADC_AWD_CH_TEMPSENSOR_ADC1_REG

ADC analog watchdog monitoring of ADC1 internal channel connected to Temperature sensor, converted by group regular only

LL_ADC_AWD_CH_TEMPSENSOR_ADC1_INJ

ADC analog watchdog monitoring of ADC1 internal channel connected to Temperature sensor, converted by group injected only

LL_ADC_AWD_CH_TEMPSENSOR_ADC1_REG_INJ

ADC analog watchdog monitoring of ADC1 internal channel connected to Temperature sensor, converted by either group regular or injected

LL_ADC_AWD_CH_TEMPSENSOR_ADC5_REG

ADC analog watchdog monitoring of ADC5 internal channel connected to Temperature sensor, converted by group regular only

LL_ADC_AWD_CH_TEMPSENSOR_ADC5_INJ

ADC analog watchdog monitoring of ADC5 internal channel connected to Temperature sensor, converted by group injected only

LL_ADC_AWD_CH_TEMPSENSOR_ADC5_REG_INJ

ADC analog watchdog monitoring of ADC5 internal channel connected to Temperature sensor, converted by either group regular or injected

LL_ADC_AWD_CH_VBAT_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda, converted by group regular only

LL_ADC_AWD_CH_VBAT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda, converted by group injected only

LL_ADC_AWD_CH_VBAT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda

LL_ADC_AWD_CH_VOPAMP1_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP1 output, channel specific to ADC1, converted by group regular only

LL_ADC_AWD_CH_VOPAMP1_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP1 output, channel specific to ADC1, converted by group injected only

LL_ADC_AWD_CH_VOPAMP1_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP1 output, channel specific to ADC1, converted by either group regular or injected

LL_ADC_AWD_CH_VOPAMP2_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP2 output, channel specific to ADC2, converted by group regular only

LL_ADC_AWD_CH_VOPAMP2_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP2 output, channel specific to ADC2, converted by group injected only

LL_ADC_AWD_CH_VOPAMP2_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP2 output, channel specific to ADC2, converted by either group regular or injected

LL_ADC_AWD_CH_VOPAMP3_ADC2_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP3 output, channel specific to ADC2, converted by group regular only

LL_ADC_AWD_CH_VOPAMP3_ADC2_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP3 output, channel specific to ADC2, converted by group injected only

LL_ADC_AWD_CH_VOPAMP3_ADC2_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP3 output, channel specific to ADC2, converted by either group regular or injected

LL_ADC_AWD_CH_VOPAMP3_ADC3_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP3 output, channel specific to ADC3, converted by group regular only

LL_ADC_AWD_CH_VOPAMP3_ADC3_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP3 output, channel specific to ADC3, converted by group injected only

LL_ADC_AWD_CH_VOPAMP3_ADC3_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP3 output, channel specific to ADC3, converted by either group regular or injected

LL_ADC_AWD_CH_VOPAMP4_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP4 output, channel specific to ADC5, converted by group regular only

LL_ADC_AWD_CH_VOPAMP4_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP4 output, channel specific to ADC5, converted by group injected only

LL_ADC_AWD_CH_VOPAMP4_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP4 output, channel specific to ADC5, converted by either group regular or injected

LL_ADC_AWD_CH_VOPAMP5_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP5 output, channel specific to ADC5, converted by group regular only

LL_ADC_AWD_CH_VOPAMP5_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP5 output, channel specific to ADC5, converted by group injected only

LL_ADC_AWD_CH_VOPAMP5_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP5 output, channel specific to ADC5, converted by either group regular or injected

LL_ADC_AWD_CH_VOPAMP6_REG

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP6 output, channel specific to ADC4, converted by group regular only

LL_ADC_AWD_CH_VOPAMP6_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP6 output, channel specific to ADC4, converted by group injected only

LL_ADC_AWD_CH_VOPAMP6_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to OPAMP6 output, channel specific to ADC4, converted by either group regular or injected

Analog watchdog - filtering config

LL_ADC_AWD_FILTERING_NONE

ADC analog wathdog no filtering, one out-of-window sample is needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_2SAMPLES

ADC analog wathdog 2 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_3SAMPLES

ADC analog wathdog 3 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_4SAMPLES

ADC analog wathdog 4 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_5SAMPLES

ADC analog wathdog 5 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_6SAMPLES

ADC analog wathdog 6 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_7SAMPLES

ADC analog wathdog 7 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_8SAMPLES

ADC analog wathdog 8 consecutives out-of-window samples are needed to raise flag or interrupt

Analog watchdog - Analog watchdog number

LL_ADC_AWD1

ADC analog watchdog number 1

LL_ADC_AWD2

ADC analog watchdog number 2

LL_ADC_AWD3

ADC analog watchdog number 3

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH

ADC analog watchdog threshold high

LL_ADC_AWD_THRESHOLD_LOW

ADC analog watchdog threshold low

LL_ADC_AWD_THRESHOLDS_HIGH_LOW

ADC analog watchdog both thresholds high and low concatenated into the same data

ADC instance - Channel number

LL_ADC_CHANNEL_0

ADC external channel (channel connected to GPIO pin) ADCx_IN0

LL_ADC_CHANNEL_1

ADC external channel (channel connected to GPIO pin) ADCx_IN1

LL_ADC_CHANNEL_2

ADC external channel (channel connected to GPIO pin) ADCx_IN2

LL_ADC_CHANNEL_3

ADC external channel (channel connected to GPIO pin) ADCx_IN3

LL_ADC_CHANNEL_4

ADC external channel (channel connected to GPIO pin) ADCx_IN4

LL_ADC_CHANNEL_5

ADC external channel (channel connected to GPIO pin) ADCx_IN5

LL_ADC_CHANNEL_6

ADC external channel (channel connected to GPIO pin) ADCx_IN6

LL_ADC_CHANNEL_7

ADC external channel (channel connected to GPIO pin) ADCx_IN7

LL_ADC_CHANNEL_8

ADC external channel (channel connected to GPIO pin) ADCx_IN8

LL_ADC_CHANNEL_9

ADC external channel (channel connected to GPIO pin) ADCx_IN9

LL_ADC_CHANNEL_10

ADC external channel (channel connected to GPIO pin) ADCx_IN10

LL_ADC_CHANNEL_11

ADC external channel (channel connected to GPIO pin) ADCx_IN11

LL_ADC_CHANNEL_12

ADC external channel (channel connected to GPIO pin) ADCx_IN12

LL_ADC_CHANNEL_13

ADC external channel (channel connected to GPIO pin) ADCx_IN13

LL_ADC_CHANNEL_14

ADC external channel (channel connected to GPIO pin) ADCx_IN14

LL_ADC_CHANNEL_15

ADC external channel (channel connected to GPIO pin) ADCx_IN15

LL_ADC_CHANNEL_16

ADC external channel (channel connected to GPIO pin) ADCx_IN16

LL_ADC_CHANNEL_17

ADC external channel (channel connected to GPIO pin) ADCx_IN17

LL_ADC_CHANNEL_18

ADC external channel (channel connected to GPIO pin) ADCx_IN18

LL_ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. On this STM32 serie, ADC channel available on all instances but ADC2.

LL_ADC_CHANNEL_TEMPSENSOR_ADC1

ADC internal channel connected to Temperature sensor. On this STM32 serie, ADC channel available only on ADC1 instance.

LL_ADC_CHANNEL_TEMPSENSOR_ADC5

ADC internal channel connected to Temperature sensor. On this STM32 serie, ADC channel available only on ADC5 instance. Refer to device datasheet for ADC5 availaibility

LL_ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/3 to have Vbat always below Vdda. On this STM32 serie, ADC channel available on all ADC instances but ADC2 & ADC4. Refer to device datasheet for ADC4 availaibility

LL_ADC_CHANNEL_VOPAMP1

ADC internal channel connected to OPAMP1 output. On this STM32 serie, ADC channel available only on ADC1 instance.

LL_ADC_CHANNEL_VOPAMP2

ADC internal channel connected to OPAMP2 output. On this STM32 serie, ADC channel available only on ADC2 instance.

LL_ADC_CHANNEL_VOPAMP3_ADC2

ADC internal channel connected to OPAMP3 output. On this STM32 serie, ADC channel available only on ADC2 instance.

LL_ADC_CHANNEL_VOPAMP3_ADC3

ADC internal channel connected to OPAMP3 output. On this STM32 serie, ADC channel available only on ADC3 instance. Refer to device datasheet for ADC3 availability

LL_ADC_CHANNEL_VOPAMP4

ADC internal channel connected to OPAMP4 output. On this STM32 serie, ADC channel available only on ADC5 instance. Refer to device datasheet for ADC5 & OPAMP4 availability

LL_ADC_CHANNEL_VOPAMP5

ADC internal channel connected to OPAMP5 output. On this STM32 serie, ADC channel available only on ADC5 instance. Refer to device datasheet for ADC5 & OPAMP5 availability

LL_ADC_CHANNEL_VOPAMP6

ADC internal channel connected to OPAMP6 output. On this STM32 serie, ADC channel available only on ADC4 instance. Refer to device datasheet for ADC4 & OPAMP6 availability

Channel - Sampling time

LL_ADC_SAMPLINGTIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_6CYCLES_5

Sampling time 6.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_24CYCLES_5

Sampling time 24.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_47CYCLES_5

Sampling time 47.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_92CYCLES_5

Sampling time 92.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_247CYCLES_5

Sampling time 247.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_640CYCLES_5

Sampling time 640.5 ADC clock cycles

Channel - Single or differential ending

LL_ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

LL_ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

LL_ADC_BOTH_SINGLE_DIFF_ENDED

ADC channel ending set to both single ended and differential (literal used only to set calibration factors)

ADC common - Clock source

LL_ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock without prescaler

LL_ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock with prescaler division by 2

LL_ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

LL_ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

LL_ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

LL_ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

LL_ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

LL_ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

LL_ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

LL_ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

LL_ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

LL_ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

LL_ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE

ADC measurement pathes all disabled

LL_ADC_PATH_INTERNAL_VREFINT

ADC measurement path to internal channel VrefInt

LL_ADC_PATH_INTERNAL_TEMPSENSOR

ADC measurement path to internal channel temperature sensor

LL_ADC_PATH_INTERNAL_VBAT

ADC measurement path to internal channel Vbat

ADC instance - Data alignment

LL_ADC_DATA_ALIGN_RIGHT

ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)

LL_ADC_DATA_ALIGN_LEFT

ADC conversion data alignment: left aligned (aligment on data register MSB bit 15)

ADC flags

LL_ADC_FLAG_ADRDY

ADC flag ADC instance ready

LL_ADC_FLAG_EOC

ADC flag ADC group regular end of unitary conversion

LL_ADC_FLAG_EOS

ADC flag ADC group regular end of sequence conversions

LL_ADC_FLAG_OVR

ADC flag ADC group regular overrun

LL_ADC_FLAG_EOSMP

ADC flag ADC group regular end of sampling phase

LL_ADC_FLAG_JEOC

ADC flag ADC group injected end of unitary conversion

LL_ADC_FLAG_JEOS

ADC flag ADC group injected end of sequence conversions

LL_ADC_FLAG_JQOVF

ADC flag ADC group injected contexts queue overflow

LL_ADC_FLAG_AWD1

ADC flag ADC analog watchdog 1

LL_ADC_FLAG_AWD2

ADC flag ADC analog watchdog 2

LL_ADC_FLAG_AWD3

ADC flag ADC analog watchdog 3

LL_ADC_FLAG_ADRDY_MST

ADC flag ADC multimode master instance ready

LL_ADC_FLAG_ADRDY_SLV

ADC flag ADC multimode slave instance ready

LL_ADC_FLAG_EOC_MST

ADC flag ADC multimode master group regular end of unitary conversion

LL_ADC_FLAG_EOC_SLV

ADC flag ADC multimode slave group regular end of unitary conversion

LL_ADC_FLAG_EOS_MST

ADC flag ADC multimode master group regular end of sequence conversions

LL_ADC_FLAG_EOS_SLV

ADC flag ADC multimode slave group regular end of sequence conversions

LL_ADC_FLAG_OVR_MST

ADC flag ADC multimode master group regular overrun

LL_ADC_FLAG_OVR_SLV

ADC flag ADC multimode slave group regular overrun

LL_ADC_FLAG_EOSMP_MST

ADC flag ADC multimode master group regular end of sampling phase

LL_ADC_FLAG_EOSMP_SLV

ADC flag ADC multimode slave group regular end of sampling phase

LL_ADC_FLAG_JEOC_MST

ADC flag ADC multimode master group injected end of unitary conversion

LL_ADC_FLAG_JEOC_SLV

ADC flag ADC multimode slave group injected end of unitary conversion

LL_ADC_FLAG_JEOS_MST

ADC flag ADC multimode master group injected end of sequence conversions

LL_ADC_FLAG_JEOS_SLV

ADC flag ADC multimode slave group injected end of sequence conversions

LL_ADC_FLAG_JQOVF_MST

ADC flag ADC multimode master group injected contexts queue overflow

LL_ADC_FLAG_JQOVF_SLV

ADC flag ADC multimode slave group injected contexts queue overflow

LL_ADC_FLAG_AWD1_MST

ADC flag ADC multimode master analog watchdog 1 of the ADC master

LL_ADC_FLAG_AWD1_SLV

ADC flag ADC multimode slave analog watchdog 1 of the ADC slave

LL_ADC_FLAG_AWD2_MST

ADC flag ADC multimode master analog watchdog 2 of the ADC master

LL_ADC_FLAG_AWD2_SLV

ADC flag ADC multimode slave analog watchdog 2 of the ADC slave

LL_ADC_FLAG_AWD3_MST

ADC flag ADC multimode master analog watchdog 3 of the ADC master

LL_ADC_FLAG_AWD3_SLV

ADC flag ADC multimode slave analog watchdog 3 of the ADC slave

ADC instance - Groups

LL_ADC_GROUP_REGULAR

ADC group regular (available on all STM32 devices)

LL_ADC_GROUP_INJECTED

ADC group injected (not available on all STM32 devices)

LL_ADC_GROUP_REGULAR_INJECTED

ADC both groups regular and injected

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US

Delay for ADC stabilization time (ADC voltage regulator start-up time)

LL_ADC_DELAY_VREFINT_STAB_US

Delay for internal voltage reference stabilization time

LL_ADC_DELAY_TEMPSENSOR_STAB_US

Delay for temperature sensor stabilization time

LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES

Delay required between ADC end of calibration and ADC enable

ADC group injected - Context queue mode

LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE

LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

LL_ADC_INJ_QUEUE_DISABLE

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE

ADC group injected sequencer discontinuous mode disable

LL_ADC_INJ_SEQ_DISCONT_1RANK

ADC group injected sequencer discontinuous mode enable with sequence interruption every rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1

ADC group injected sequencer rank 1

LL_ADC_INJ_RANK_2

ADC group injected sequencer rank 2

LL_ADC_INJ_RANK_3

ADC group injected sequencer rank 3

LL_ADC_INJ_RANK_4

ADC group injected sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE

ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS

ADC group injected sequencer enable with 2 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS

ADC group injected sequencer enable with 3 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING

ADC group injected conversion trigger polarity set to rising edge

LL_ADC_INJ_TRIG_EXT_FALLING

ADC group injected conversion trigger polarity set to falling edge

LL_ADC_INJ_TRIG_EXT_RISINGFALLING

ADC group injected conversion trigger polarity set to both rising and falling edges

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE

ADC group injected conversion trigger internal: SW start.. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_CH3

ADC group injected conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availability

LL_ADC_INJ_TRIG_EXT_TIM1_CH4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_CH1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_INJ_TRIG_EXT_TIM3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_INJ_TRIG_EXT_TIM3_CH3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_INJ_TRIG_EXT_TIM3_CH4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_INJ_TRIG_EXT_TIM4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM4_CH3

ADC group injected conversion trigger from external peripheral: TIM4 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availability

LL_ADC_INJ_TRIG_EXT_TIM4_CH4

ADC group injected conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_INJ_TRIG_EXT_TIM6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM7_TRGO

ADC group injected conversion trigger from external peripheral: TIM7 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_CH2

ADC group injected conversion trigger from external peripheral: TIM8 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_INJ_TRIG_EXT_TIM8_CH4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM16_CH1

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_INJ_TRIG_EXT_TIM20_TRGO

ADC group injected conversion trigger from external peripheral: TIM20 TRGO. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_TIM20_TRGO2

ADC group injected conversion trigger from external peripheral: TIM20 TRGO2. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_TIM20_CH2

ADC group injected conversion trigger from external peripheral: TIM20 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Trigger available only on ADC3/4/5 instances. On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_TIM20_CH4

ADC group injected conversion trigger from external peripheral: TIM20 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Trigger available only on ADC1/2 instances. On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG1

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 1 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances, and HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 2 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG3

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 3 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances, and HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 4 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG5

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 5 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG6

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 6 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG7

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 7 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG8

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 8 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG9

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 9 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG10

ADC group injected conversion trigger from external peripheral: HRTIMER ADC trigger 10 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_INJ_TRIG_EXT_EXTI_LINE3

ADC group injected conversion trigger from external peripheral: external interrupt line 3. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances.

LL_ADC_INJ_TRIG_EXT_LPTIM_OUT

ADC group injected conversion trigger from external peripheral: LPTIMER OUT event. Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT

ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.

LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY

ADC interruption ADC instance ready

LL_ADC_IT_EOC

ADC interruption ADC group regular end of unitary conversion

LL_ADC_IT_EOS

ADC interruption ADC group regular end of sequence conversions

LL_ADC_IT_OVR

ADC interruption ADC group regular overrun

LL_ADC_IT_EOSMP

ADC interruption ADC group regular end of sampling phase

LL_ADC_IT_JEOC

ADC interruption ADC group injected end of unitary conversion

LL_ADC_IT_JEOS

ADC interruption ADC group injected end of sequence conversions

LL_ADC_IT_JQOVF

ADC interruption ADC group injected contexts queue overflow

LL_ADC_IT_AWD1

ADC interruption ADC analog watchdog 1

LL_ADC_IT_AWD2

ADC interruption ADC analog watchdog 2

LL_ADC_IT_AWD3

ADC interruption ADC analog watchdog 3

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE

No ADC low power mode activated

LL_ADC_LP_AUTOWAIT

ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function

Multimode - DMA transfer

LL_ADC_MULTI_REG_DMA_EACH_ADC

ADC multimode group regular conversions are transferred by DMA: each ADC uses its own DMA channel, with its individual DMA transfer settings

LL_ADC_MULTI_REG_DMA_LIMIT_RES12_10B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master), in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting for ADC resolution of 12 and 10 bits

LL_ADC_MULTI_REG_DMA_LIMIT_RES8_6B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master), in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting for ADC resolution of 8 and 6 bits

LL_ADC_MULTI_REG_DMA_UNLMT_RES12_10B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. Setting for ADC resolution of 12 and 10 bits

LL_ADC_MULTI_REG_DMA_UNLMT_RES8_6B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. Setting for ADC resolution of 8 and 6 bits

Multimode - ADC master or slave

LL_ADC_MULTI_MASTER

In multimode, selection among several ADC instances: ADC master

LL_ADC_MULTI_SLAVE

In multimode, selection among several ADC instances: ADC slave

LL_ADC_MULTI_MASTER_SLAVE

In multimode, selection among several ADC instances: both ADC master and ADC slave

Multimode - Mode

LL_ADC_MULTI_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

LL_ADC_MULTI_DUAL_REG_SIMULT

ADC dual mode enabled: group regular simultaneous

LL_ADC_MULTI_DUAL_REG_INTERL

ADC dual mode enabled: Combined group regular interleaved

LL_ADC_MULTI_DUAL_INJ_SIMULT

ADC dual mode enabled: group injected simultaneous

LL_ADC_MULTI_DUAL_INJ_ALTERN

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles

ADC instance - Offset number

LL_ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC instance - Offset saturation mode

LL_ADC_OFFSET_SATURATION_DISABLE

ADC offset saturation is disabled (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_SATURATION_ENABLE

ADC offset saturation is enabled (among ADC selected offset number 1, 2, 3 or 4)

ADC instance - Offset sign

LL_ADC_OFFSET_SIGN_NEGATIVE

ADC offset is negative (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_SIGN_POSITIVE

ADC offset is positive (among ADC selected offset number 1, 2, 3 or 4)

ADC instance - Offset state

LL_ADC_OFFSET_DISABLE

ADC offset disabled (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_ENABLE

ADC offset enabled (among ADC selected offset number 1, 2, 3 or 4)

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

LL_ADC_OVS_REG_DISCONT

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

LL_ADC_OVS_RATIO_2

ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_4

ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_8

ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_16

ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_32

ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_64

ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_128

ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_256

ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE

ADC oversampling disabled.

LL_ADC_OVS_GRP_REGULAR_CONTINUED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is temporary stopped and continued afterwards.

LL_ADC_OVS_GRP_REGULAR_RESUMED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

LL_ADC_OVS_GRP_INJECTED

ADC oversampling on conversions of ADC group injected.

LL_ADC_OVS_GRP_INJ_REG_RESUMED

ADC oversampling on conversions of both ADC groups regular and injected. If group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

Oversampling - Data shift

LL_ADC_OVS_SHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_1

ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_2

ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_3

ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_4

ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_5

ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_6

ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_7

ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_8

ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

LL_ADC_DMA_REG_REGULAR_DATA_MULTI

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE

ADC conversions are performed in single mode: one conversion per trigger

LL_ADC_REG_CONV_CONTINUOUS

ADC conversions are performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE

ADC conversions are not transferred by DMA

LL_ADC_REG_DMA_TRANSFER_LIMITED

ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRANSFER_UNLIMITED

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sampling mode

LL_ADC_REG_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

LL_ADC_REG_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: Trigger rising edge = start sampling Trigger falling edge = stop sampling and start conversion

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE

ADC group regular sequencer discontinuous mode disable

LL_ADC_REG_SEQ_DISCONT_1RANK

ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

LL_ADC_REG_SEQ_DISCONT_2RANKS

ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks

LL_ADC_REG_SEQ_DISCONT_3RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks

LL_ADC_REG_SEQ_DISCONT_4RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks

LL_ADC_REG_SEQ_DISCONT_5RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks

LL_ADC_REG_SEQ_DISCONT_6RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks

LL_ADC_REG_SEQ_DISCONT_7RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks

LL_ADC_REG_SEQ_DISCONT_8RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1

ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2

ADC group regular sequencer rank 2

LL_ADC_REG_RANK_3

ADC group regular sequencer rank 3

LL_ADC_REG_RANK_4

ADC group regular sequencer rank 4

LL_ADC_REG_RANK_5

ADC group regular sequencer rank 5

LL_ADC_REG_RANK_6

ADC group regular sequencer rank 6

LL_ADC_REG_RANK_7

ADC group regular sequencer rank 7

LL_ADC_REG_RANK_8

ADC group regular sequencer rank 8

LL_ADC_REG_RANK_9

ADC group regular sequencer rank 9

LL_ADC_REG_RANK_10

ADC group regular sequencer rank 10

LL_ADC_REG_RANK_11

ADC group regular sequencer rank 11

LL_ADC_REG_RANK_12

ADC group regular sequencer rank 12

LL_ADC_REG_RANK_13

ADC group regular sequencer rank 13

LL_ADC_REG_RANK_14

ADC group regular sequencer rank 14

LL_ADC_REG_RANK_15

ADC group regular sequencer rank 15

LL_ADC_REG_RANK_16

ADC group regular sequencer rank 16

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE

ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS

ADC group regular sequencer enable with 2 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS

ADC group regular sequencer enable with 3 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS

ADC group regular sequencer enable with 4 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS

ADC group regular sequencer enable with 5 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS

ADC group regular sequencer enable with 6 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS

ADC group regular sequencer enable with 7 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS

ADC group regular sequencer enable with 8 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS

ADC group regular sequencer enable with 9 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS

ADC group regular sequencer enable with 10 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS

ADC group regular sequencer enable with 11 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS

ADC group regular sequencer enable with 12 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS

ADC group regular sequencer enable with 13 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS

ADC group regular sequencer enable with 14 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS

ADC group regular sequencer enable with 15 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

ADC group regular sequencer enable with 16 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING

ADC group regular conversion trigger polarity set to rising edge

LL_ADC_REG_TRIG_EXT_FALLING

ADC group regular conversion trigger polarity set to falling edge

LL_ADC_REG_TRIG_EXT_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE

ADC group regular conversion trigger internal: SW start.

LL_ADC_REG_TRIG_EXT_TIM1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_REG_TRIG_EXT_TIM1_CH2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_REG_TRIG_EXT_TIM1_CH3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_CH1

ADC group regular conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_REG_TRIG_EXT_TIM2_CH2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_REG_TRIG_EXT_TIM2_CH3

ADC group regular conversion trigger from external peripheral: TIM2 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_REG_TRIG_EXT_TIM3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_CH1

ADC group regular conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_REG_TRIG_EXT_TIM3_CH4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_REG_TRIG_EXT_TIM4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_CH1

ADC group regular conversion trigger from external peripheral: TIM4 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_REG_TRIG_EXT_TIM4_CH4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_REG_TRIG_EXT_TIM6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM7_TRGO

ADC group regular conversion trigger from external peripheral: TIM7 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_CH1

ADC group regular conversion trigger from external peripheral: TIM8 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_REG_TRIG_EXT_TIM15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM20_TRGO

ADC group regular conversion trigger from external peripheral: TIM20 TRGO. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_TIM20_TRGO2

ADC group regular conversion trigger from external peripheral: TIM20 TRGO2. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_TIM20_CH1

ADC group regular conversion trigger from external peripheral: TIM20 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_TIM20_CH2

ADC group regular conversion trigger from external peripheral: TIM20 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances, and TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_TIM20_CH3

ADC group regular conversion trigger from external peripheral: TIM20 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances, and TIM20 is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG1

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 1 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG2

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 2 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances, and HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG3

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 3 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG4

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 4 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances, and HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG5

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 5 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG6

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 6 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG7

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 7 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG8

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 8 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG9

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 9 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_HRTIM_TRG10

ADC group regular conversion trigger from external peripheral: HRTIMER ADC trigger 10 event. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, HRTIM is not available on all devices. Refer to device datasheet for more details

LL_ADC_REG_TRIG_EXT_EXTI_LINE11

ADC group regular conversion trigger from external peripheral: external interrupt line 11. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC1/2 instances

LL_ADC_REG_TRIG_EXT_EXTI_LINE2

ADC group regular conversion trigger from external peripheral: external interrupt line 2. Trigger edge set to rising edge (default setting). Note: On this STM32 serie, this trigger is available only on ADC3/4/5 instances. Refer to device datasheet for ADCx availaibility

LL_ADC_REG_TRIG_EXT_LPTIM_OUT

ADC group regular conversion trigger from external peripheral: LPTIMER OUT event. Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_12B

ADC resolution 12 bits

LL_ADC_RESOLUTION_10B

ADC resolution 10 bits

LL_ADC_RESOLUTION_8B

ADC resolution 8 bits

LL_ADC_RESOLUTION_6B

ADC resolution 6 bits

ADC instance - ADC sampling time common configuration

LL_ADC_SAMPLINGTIME_COMMON_DEFAULT

ADC sampling time let to default settings.

LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5

ADC additional sampling time 3.5 ADC clock cycles replacing 2.5 ADC clock cycles (this applies to all channels mapped with selection sampling time 2.5 ADC clock cycles, whatever channels mapped on ADC groups regular or injected).

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution. (1, 2, 3, 4, 5, 7) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution.

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details.

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1 (8)
  - LL_ADC_CHANNEL_2 (8)
  - LL_ADC_CHANNEL_3 (8)
  - LL_ADC_CHANNEL_4 (8)
  - LL_ADC_CHANNEL_5 (8)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT (7)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC1 (1)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC5 (5)
  - LL_ADC_CHANNEL_VBAT (6)
  - LL_ADC_CHANNEL_VOPAMP1 (1)
  - LL_ADC_CHANNEL_VOPAMP2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC2 (2)
  - LL_ADC_CHANNEL_VOPAMP3_ADC3 (3)
  - LL_ADC_CHANNEL_VOPAMP4 (5)
  - LL_ADC_CHANNEL_VOPAMP5 (5)
  - LL_ADC_CHANNEL_VOPAMP6 (4)
  - On this STM32 serie, all ADCx are not available on all devices. Refer to device datasheet for more details. (8) On STM32G4, fast channel allows: 2.5 (sampling) + 12.5 (conversion) = 15 ADC clock cycles (fADC) to convert in 12-bit resolution. Other channels are slow channels allows: 6.5 (sampling) + 12.5 (conversion) = 19 ADC clock cycles (fADC) to convert in 12-bit resolution. (1, 2, 3, 4, 5, 7) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ
  - LL_ADC_AWD_CHANNEL_15_REG (0)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example: LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1, __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_ConfigAnalogWDThresholds() or LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param >, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD_12_BITS__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_6_bits > = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW

Description:

- Helper macro to get the ADC analog watchdog threshold high or low from raw value containing both thresholds concatenated.

Parameters:

- __AWD_THRESHOLD_TYPE__: This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- __AWD_THRESHOLDS__: Value between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, to get analog watchdog threshold high from the register raw value: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW(LL_ADC_AWD_THRESHOLD_HIGH, <raw_value_with_both_thresholds>);

__LL_ADC_CALIB_FACTOR_SINGLE_DIFF

Description:

- Helper macro to set the ADC calibration value with both single ended and differential modes calibration factors concatenated.

Parameters:

- __CALIB_FACTOR_SINGLE_ENDED__: Value between Min_Data=0x00 and Max_Data=0x7F
- __CALIB_FACTOR_DIFFERENTIAL__: Value between Min_Data=0x00 and Max_Data=0x7F

Return value:

- Value: between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes:

- To be used with function LL_ADC_SetCalibrationFactor(). Example, to set calibration factors single ended to 0x55 and differential ended to 0x2A: LL_ADC_SetCalibrationFactor(ADC1, __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(0x55, 0x2A))

__LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__LL_ADC_MULTI_INSTANCE_MASTER

Description:

- Helper macro to select, from a ADC instance, to which ADC instance it has a dependence in multimode (ADC master of the corresponding ADC common instance).

Parameters:

- __ADCx__: ADC instance

Return value:

- __ADCx__: ADC instance master of the corresponding ADC common instance

Notes:

- In case of device with multimode available and a mix of ADC instances compliant and not compliant with multimode feature, ADC instances not compliant with multimode feature are considered as master instances (do not depend to any other ADC instance).

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value (unit: digital value of ADC conversion data)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref + and Vref- (refer to reference manual).

__LL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 serie, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__LL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 serie, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__LL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32G4, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32G4, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12 bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

66 LL BUS Generic Driver

66.1 BUS Firmware driver API description

The following section lists the various functions of the BUS library.

66.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_EnableClock
- AHB1ENR DMAMMUXEN LL_AHB1_GRP1_EnableClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_EnableClock
- AHB1ENR FMACEN LL_AHB1_GRP1_EnableClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_EnableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_EnableClock

LL_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMAMUXEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FMACEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CRCEN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMAMUXEN LL_AHB1_GRP1_DisableClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_DisableClock
- AHB1ENR FMACEN LL_AHB1_GRP1_DisableClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_DisableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR DMA1RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMA2RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMAMUXRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR CORDICRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR FMACRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR FLASHRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR CRCRST LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR DMA1RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMA2RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMAMUXRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CORDICRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR FMACRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR FLASHRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CRCRST LL_AHB1_GRP1_ReleaseReset

LL_AHB1_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_SRAM1
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1SMENR DMA1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DMA2SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DMAMUXSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR CORDICSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR FMACSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_EnableClockStopSleep

LL_AHB1_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_DMAMUX1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_SRAM1
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1SMENR DMA1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DMA2SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DMAMUXSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR CORDICSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR FMACSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_DisableClockStopSleep

LL_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR GPIOAEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOBEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOCEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIODEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOEEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOFEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR ADC12EN LL_AHB2_GRP1_EnableClock
- AHB2ENR ADC345EN LL_AHB2_GRP1_EnableClock
- AHB2ENR DAC1EN LL_AHB2_GRP1_EnableClock
- AHB2ENR DAC2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR DAC3EN LL_AHB2_GRP1_EnableClock
- AHB2ENR DAC4EN LL_AHB2_GRP1_EnableClock
- AHB2ENR AESEN LL_AHB2_GRP1_EnableClock
- AHB2ENR RNGEN LL_AHB2_GRP1_EnableClock

LL_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2ENR GPIOAEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOBEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOCEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIODEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOEEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOFEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR ADC12EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR ADC345EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR DAC1EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR DAC2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR DAC3EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR DAC4EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR AESEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR RNGEN LL_AHB2_GRP1_IsEnabledClock

LL_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR GPIOAEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOBEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOCEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIODEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOEEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOFEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR ADC12EN LL_AHB2_GRP1_DisableClock
- AHB2ENR ADC345EN LL_AHB2_GRP1_DisableClock
- AHB2ENR DAC1EN LL_AHB2_GRP1_DisableClock
- AHB2ENR DAC2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR DAC3EN LL_AHB2_GRP1_DisableClock
- AHB2ENR DAC4EN LL_AHB2_GRP1_DisableClock
- AHB2ENR AESEN LL_AHB2_GRP1_DisableClock
- AHB2ENR RNGEN LL_AHB2_GRP1_DisableClock

LL_AHB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR GPIOARST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOBRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOCRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIODRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOERST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOFRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR GPIOGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR ADC12RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR ADC345RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR DAC1RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR DAC2RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR DAC3RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR DAC4RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR AESRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR RNGRST LL_AHB2_GRP1_ForceReset

LL_AHB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR GPIOARST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR GPIOBRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR GPIOCRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR GPIODRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR GPIOERST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR GPIOFRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR GPIOGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR ADC12RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR ADC345RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR DAC1RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR DAC2RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR DAC3RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR DAC4RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR AESRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR RNGRST LL_AHB2_GRP1_ReleaseReset

LL_AHB2_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_SRAM2
  - LL_AHB2_GRP1_PERIPH_CCM
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR GPIOASMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOBSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOCSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIODSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOESMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOFSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR GPIOGSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR SRAM2SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR CCMSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR ADC12SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR ADC345SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR DAC1SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR DAC2SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR DAC3SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR DAC4SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR AESSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR RNGSMEN LL_AHB2_GRP1_EnableClockStopSleep

LL_AHB2_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_SRAM2
  - LL_AHB2_GRP1_PERIPH_CCM
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_ADC345 (*)
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DAC2 (*)
  - LL_AHB2_GRP1_PERIPH_DAC3
  - LL_AHB2_GRP1_PERIPH_DAC4 (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_RNG

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR GPIOASMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOBSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOCSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIODSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOESMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOFSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR GPIOGSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR SRAM2SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR CCMSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR ADC12SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR ADC345SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR DAC1SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR DAC2SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR DAC3SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR DAC4SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR AESSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR RNGSMEN LL_AHB2_GRP1_DisableClockStopSleep

LL_AHB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR FMCEN LL_AHB3_GRP1_EnableClock
- AHB3ENR QSPIEN LL_AHB3_GRP1_EnableClock

LL_AHB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB3ENR FMCEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR QSPIEN LL_AHB3_GRP1_IsEnabledClock

LL_AHB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR FMCEN LL_AHB3_GRP1_DisableClock
- AHB3ENR QSPIEN LL_AHB3_GRP1_DisableClock

LL_AHB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_ALL
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTR FMCRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR QSPIRST LL_AHB3_GRP1_ForceReset

LL_AHB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_ALL
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTR FMCRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR QSPIRST LL_AHB3_GRP1_ReleaseReset

LL_AHB3_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB3 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3SMENR FMCSMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR QSPISMEN LL_AHB3_GRP1_EnableClockStopSleep

LL_AHB3_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB3 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_FMC (*)
  - LL_AHB3_GRP1_PERIPH_QSPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3SMENR FMCSMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR QSPISMEN LL_AHB3_GRP1_DisableClockStopSleep

LL_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_RTCAPB
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_EnableClock
- APB1ENR1 CRSEN LL_APB1_GRP1_EnableClock
- APB1ENR1 RTCAPBEN LL_APB1_GRP1_EnableClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_EnableClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 SPI3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 UART4EN LL_APB1_GRP1_EnableClock
- APB1ENR1 UART5EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USBEN LL_APB1_GRP1_EnableClock
- APB1ENR1 FDCANEN LL_APB1_GRP1_EnableClock
- APB1ENR1 PWREN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 LPTIM1EN LL_APB1_GRP1_EnableClock

LL_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR2 LPUART1EN LL_APB1_GRP2_EnableClock
- APB1ENR2 I2C4EN LL_APB1_GRP2_EnableClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_RTCAPB
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 CRSEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 RTCAPBEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 SPI3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USBEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 FDCANEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 PWREN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 LPTIM1EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1ENR2 LPUART1EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 I2C4EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_RTCAPB
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_DisableClock
- APB1ENR1 CRSEN LL_APB1_GRP1_DisableClock
- APB1ENR1 RTCAPBEN LL_APB1_GRP1_DisableClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_DisableClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 SPI3EN LL_APB1_GRP1_DisableClock
- APB1ENR1 USART2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 USART3EN LL_APB1_GRP1_DisableClock
- APB1ENR1 UART4EN LL_APB1_GRP1_DisableClock
- APB1ENR1 UART5EN LL_APB1_GRP1_DisableClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_DisableClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 USBEN LL_APB1_GRP1_DisableClock
- APB1ENR1 FDCANEN LL_APB1_GRP1_DisableClock
- APB1ENR1 PWREN LL_APB1_GRP1_DisableClock
- APB1ENR1 I2C3EN LL_APB1_GRP1_DisableClock
- APB1ENR1 LPTIM1EN LL_APB1_GRP1_DisableClock

LL_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR2 LPUART1EN LL_APB1_GRP2_DisableClock
- APB1ENR2 I2C4EN LL_APB1_GRP2_DisableClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_DisableClock

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR1 TIM2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM4RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM5RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM6RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM7RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 CRSRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 SPI2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 SPI3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 UART4RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 UART5RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C1RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USBRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 FDCANRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 PWRRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 LPTIM1RST LL_APB1_GRP1_ForceReset

LL_APB1_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR2 LPUART1RST LL_APB1_GRP2_ForceReset
- APB1RSTR2 I2C4RST LL_APB1_GRP2_ForceReset
- APB1RSTR2 UCPD1RST LL_APB1_GRP2_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR1 TIM2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM6RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM7RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 CRSRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 SPI2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 SPI3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 USART2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 USART3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 UART4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 UART5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 I2C1RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 I2C2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 USBRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 FDCANRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 PWRRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 I2C3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 LPTIM1RST LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR2 LPUART1RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 I2C4RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 UCPD1RST LL_APB1_GRP2_ReleaseReset

LL_APB1_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_RTCAPB
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 RTCAPBSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 WWDGSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 SPI3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USBSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 FDCANSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 PWRSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 LPTIM1SMEN LL_APB1_GRP1_EnableClockStopSleep

LL_APB1_GRP2_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR2 LPUART1SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 I2C4SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 UCPD1SMEN LL_APB1_GRP2_EnableClockStopSleep

LL_APB1_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_RTCAPB
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_USB
  - LL_APB1_GRP1_PERIPH_FDCAN (*)
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_LPTIM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 RTCAPBSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 WWDGSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 SPI3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USBSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 FDCANSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 PWRSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 LPTIM1SMEN LL_APB1_GRP1_DisableClockStopSleep

LL_APB1_GRP2_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_LPUART1
  - LL_APB1_GRP2_PERIPH_I2C4 (*)
  - LL_APB1_GRP2_PERIPH_UCPD1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR2 LPUART1SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 I2C4SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 UCPD1SMEN LL_APB1_GRP2_DisableClockStopSleep

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR SYSCFGEN LL_APB2_GRP1_EnableClock
- APB2ENR TIM1EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI4EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM20EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_APB2_GRP1_EnableClock
- APB2ENR HRTIM1EN LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB2ENR SYSCFGEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI4EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM20EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR HRTIM1EN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR SYSCFGEN LL_APB2_GRP1_DisableClock
- APB2ENR TIM1EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI4EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM20EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI1EN LL_APB2_GRP1_DisableClock
- APB2ENR HRTIM1EN LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR SYSCFGRST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI4RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM20RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR HRTIM1RST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR SYSCFGRST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI4RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM20RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR HRTIM1RST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2SMENR SYSCFGSMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SPI1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM8SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR USART1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SPI4SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM15SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM16SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM17SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM20SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR HRTIM1SMEN LL_APB2_GRP1_EnableClockStopSleep

LL_APB2_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_SYSCFG
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_TIM20 (*)
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_HRTIM1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2SMENR SYSCFGSMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SPI1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM8SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR USART1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SPI4SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM15SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM16SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM17SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM20SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR HRTIM1SMEN LL_APB2_GRP1_DisableClockStopSleep

66.2 BUS Firmware driver defines

The following section lists the various define and macros of the module.

66.2.1 BUS

BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_DMA1

LL_AHB1_GRP1_PERIPH_DMA2

LL_AHB1_GRP1_PERIPH_DMAMUX1

LL_AHB1_GRP1_PERIPH_CORDIC

LL_AHB1_GRP1_PERIPH_FMAC

LL_AHB1_GRP1_PERIPH_FLASH

LL_AHB1_GRP1_PERIPH_SRAM1

LL_AHB1_GRP1_PERIPH_CRC

AHB2 GRP1 PERIPH

LL_AHB2_GRP1_PERIPH_ALL

LL_AHB2_GRP1_PERIPH_GPIOA

LL_AHB2_GRP1_PERIPH_GPIOB

LL_AHB2_GRP1_PERIPH_GPIOC

LL_AHB2_GRP1_PERIPH_GPIOD

LL_AHB2_GRP1_PERIPH_GPIOE

LL_AHB2_GRP1_PERIPH_GPIOF

LL_AHB2_GRP1_PERIPH_GPIOG

LL_AHB2_GRP1_PERIPH_CCM

LL_AHB2_GRP1_PERIPH_SRAM2

LL_AHB2_GRP1_PERIPH_ADC12

LL_AHB2_GRP1_PERIPH_ADC345

LL_AHB2_GRP1_PERIPH_DAC1

LL_AHB2_GRP1_PERIPH_DAC2

LL_AHB2_GRP1_PERIPH_DAC3

LL_AHB2_GRP1_PERIPH_DAC4

LL_AHB2_GRP1_PERIPH_AES

LL_AHB2_GRP1_PERIPH_RNG

AHB3 GRP1 PERIPH

LL_AHB3_GRP1_PERIPH_ALL

LL_AHB3_GRP1_PERIPH_FMC

LL_AHB3_GRP1_PERIPH_QSPI

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_CRS

LL_APB1_GRP1_PERIPH_RTCAPB

LL_APB1_GRP1_PERIPH_WWDG

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_USB

LL_APB1_GRP1_PERIPH_FDCAN

LL_APB1_GRP1_PERIPH_PWR

LL_APB1_GRP1_PERIPH_I2C3

LL_APB1_GRP1_PERIPH_LPTIM1

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_ALL

LL_APB1_GRP2_PERIPH_LPUART1

LL_APB1_GRP2_PERIPH_I2C4

LL_APB1_GRP2_PERIPH_UCPD1

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_ALL

LL_APB2_GRP1_PERIPH_SYSCFG

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_TIM8

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_SPI4

LL_APB2_GRP1_PERIPH_TIM15

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_TIM20

LL_APB2_GRP1_PERIPH_SAI1

LL_APB2_GRP1_PERIPH_HRTIM1

67 LL COMP Generic Driver

67.1 COMP Firmware driver registers structures

67.1.1 LL_COMP_InitTypeDef

LL_COMP_InitTypeDef is defined in the stm32g4xx_ll_comp.h

Data Fields

- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t InputHysteresis***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputBlankingSource***

Field Documentation

- ***uint32_t LL_COMP_InitTypeDef::InputPlus***

  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_PLUS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputPlus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputMinus***

  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_MINUS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputMinus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputHysteresis***

  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_LL_EC_INPUT_HYSTERESIS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputHysteresis()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputPolarity***

  Set comparator output polarity. This parameter can be a value of ***COMP_LL_EC_OUTPUT_POLARITY***This feature can be modified afterwards using unitary function **LL_COMP_SetOutputPolarity()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputBlankingSource***

  Set comparator blanking source. This parameter can be a value of ***COMP_LL_EC_OUTPUT_BLANKING_SOURCE***This feature can be modified afterwards using unitary function **LL_COMP_SetOutputBlankingSource()**.

67.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

67.2.1 Detailed description of functions

LL_COMP_ConfigInputs

Function name

__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)

Function description

Set comparator inputs minus (inverting) and plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1 (1,3,4)
  - LL_COMP_INPUT_MINUS_DAC1_CH2 (2,5)
  - LL_COMP_INPUT_MINUS_DAC2_CH1 (6,7)
  - LL_COMP_INPUT_MINUS_DAC3_CH1 (1,3)
  - LL_COMP_INPUT_MINUS_DAC3_CH2 (2,4)
  - LL_COMP_INPUT_MINUS_DAC4_CH1 (5,7)
  - LL_COMP_INPUT_MINUS_DAC4_CH2 (6) (a,b...) Only available for COMPa, COMPb... For COMPx & DACx instances availability, please refer to datasheet
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
- **InputPlus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 serie, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.

Reference Manual to LL API cross reference:

- CSR INMSEL LL_COMP_ConfigInputs
- CSR INPSEL LL_COMP_ConfigInputs
- CSR BRGEN LL_COMP_ConfigInputs
- CSR SCALEN LL_COMP_ConfigInputs

LL_COMP_SetInputPlus

Function name

__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)

Function description

Set comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputPlus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR INPSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (COMP_TypeDef * COMPx)

Function description

Get comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR INPSEL LL_COMP_GetInputPlus

LL_COMP_SetInputMinus

Function name

__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)

Function description

Set comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1 (1,3,4)
  - LL_COMP_INPUT_MINUS_DAC1_CH2 (2,5)
  - LL_COMP_INPUT_MINUS_DAC2_CH1 (6,7)
  - LL_COMP_INPUT_MINUS_DAC3_CH1 (1,3)
  - LL_COMP_INPUT_MINUS_DAC3_CH2 (2,4)
  - LL_COMP_INPUT_MINUS_DAC4_CH1 (5,7)
  - LL_COMP_INPUT_MINUS_DAC4_CH2 (6) (a,b...) Only available for COMPa, COMPb... For COMPx & DACx instances availability, please refer to datasheet
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 serie, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.

Reference Manual to LL API cross reference:

- CSR INMSEL LL_COMP_SetInputMinus
- CSR BRGEN LL_COMP_SetInputMinus
- CSR SCALEN LL_COMP_SetInputMinus

LL_COMP_GetInputMinus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputMinus (COMP_TypeDef * COMPx)

Function description

Get comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1 (1,3,4)
  - LL_COMP_INPUT_MINUS_DAC1_CH2 (2,5)
  - LL_COMP_INPUT_MINUS_DAC2_CH1 (6,7)
  - LL_COMP_INPUT_MINUS_DAC3_CH1 (1,3)
  - LL_COMP_INPUT_MINUS_DAC3_CH2 (2,4)
  - LL_COMP_INPUT_MINUS_DAC4_CH1 (5,7)
  - LL_COMP_INPUT_MINUS_DAC4_CH2 (6) (a,b...) Only available for COMPa, COMPb... For COMPx & DACx instances availability, please refer to datasheet
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR INMSEL LL_COMP_GetInputMinus
- CSR BRGEN LL_COMP_GetInputMinus
- CSR SCALEN LL_COMP_GetInputMinus

LL_COMP_SetInputHysteresis

Function name

__STATIC_INLINE void LL_COMP_SetInputHysteresis (COMP_TypeDef * COMPx, uint32_t InputHysteresis)

Function description

Set comparator instance hysteresis mode of the input minus (inverting input).

Parameters

- **COMPx:** Comparator instance
- **InputHysteresis:** This parameter can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_10MV
  - LL_COMP_HYSTERESIS_20MV
  - LL_COMP_HYSTERESIS_30MV
  - LL_COMP_HYSTERESIS_40MV
  - LL_COMP_HYSTERESIS_50MV
  - LL_COMP_HYSTERESIS_60MV
  - LL_COMP_HYSTERESIS_70MV
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR HYST LL_COMP_SetInputHysteresis

LL_COMP_GetInputHysteresis

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputHysteresis (COMP_TypeDef * COMPx)

Function description

Get comparator instance hysteresis mode of the minus (inverting) input.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_10MV
  - LL_COMP_HYSTERESIS_20MV
  - LL_COMP_HYSTERESIS_30MV
  - LL_COMP_HYSTERESIS_40MV
  - LL_COMP_HYSTERESIS_50MV
  - LL_COMP_HYSTERESIS_60MV
  - LL_COMP_HYSTERESIS_70MV

Reference Manual to LL API cross reference:

- CSR HYST LL_COMP_GetInputHysteresis

LL_COMP_SetOutputPolarity

Function name

__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)

Function description

Set comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance
- **OutputPolarity:** This parameter can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR POLARITY LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (COMP_TypeDef * COMPx)

Function description

Get comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Reference Manual to LL API cross reference:

- CSR POLARITY LL_COMP_GetOutputPolarity

LL_COMP_SetOutputBlankingSource

Function name

__STATIC_INLINE void LL_COMP_SetOutputBlankingSource (COMP_TypeDef * COMPx, uint32_t BlankingSource)

Function description

Set comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance
- **BlankingSource:** This parameter can be one of the following values:
  - LL_COMP_BLANKINGSRC_NONE
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP1
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP2
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP3
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP4
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP5
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP6
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP7
  - LL_COMP_BLANKINGSRC_TIM2_OC3_COMP1
  - LL_COMP_BLANKINGSRC_TIM2_OC3_COMP2
  - LL_COMP_BLANKINGSRC_TIM2_OC3_COMP5
  - LL_COMP_BLANKINGSRC_TIM2_OC4_COMP3
  - LL_COMP_BLANKINGSRC_TIM2_OC4_COMP6
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP1
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP2
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP3
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP5
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP7
  - LL_COMP_BLANKINGSRC_TIM3_OC4_COMP4
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP1
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP2
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP3
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP4
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP5
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP6
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP7
  - LL_COMP_BLANKINGSRC_TIM15_OC1_COMP4
  - LL_COMP_BLANKINGSRC_TIM15_OC2_COMP6
  - LL_COMP_BLANKINGSRC_TIM15_OC2_COMP7
  - LL_COMP_BLANKINGSRC_TIM20_OC5
  - LL_COMP_BLANKINGSRC_TIM15_OC1
  - LL_COMP_BLANKINGSRC_TIM4_OC3

  On STM32G4 series, blanking sources are linked to COMP instance (except those without COMPx suffix that are common to all instances) Note: For COMPx & TIMx instances availability, please refer to datasheet

Return values

- **None:**

Notes

- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.
- Availability of parameters of blanking source from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CSR BLANKING LL_COMP_SetOutputBlankingSource

LL_COMP_GetOutputBlankingSource

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputBlankingSource (COMP_TypeDef * COMPx)

Function description

Get comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_BLANKINGSRC_NONE
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP1
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP2
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP3
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP4
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP5
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP6
  - LL_COMP_BLANKINGSRC_TIM1_OC5_COMP7
  - LL_COMP_BLANKINGSRC_TIM2_OC3_COMP1
  - LL_COMP_BLANKINGSRC_TIM2_OC3_COMP2
  - LL_COMP_BLANKINGSRC_TIM2_OC3_COMP5
  - LL_COMP_BLANKINGSRC_TIM2_OC4_COMP3
  - LL_COMP_BLANKINGSRC_TIM2_OC4_COMP6
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP1
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP2
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP3
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP5
  - LL_COMP_BLANKINGSRC_TIM3_OC3_COMP7
  - LL_COMP_BLANKINGSRC_TIM3_OC4_COMP4
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP1
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP2
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP3
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP4
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP5
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP6
  - LL_COMP_BLANKINGSRC_TIM8_OC5_COMP7
  - LL_COMP_BLANKINGSRC_TIM15_OC1_COMP4
  - LL_COMP_BLANKINGSRC_TIM15_OC2_COMP6
  - LL_COMP_BLANKINGSRC_TIM15_OC2_COMP7
  - LL_COMP_BLANKINGSRC_TIM20_OC5
  - LL_COMP_BLANKINGSRC_TIM15_OC1
  - LL_COMP_BLANKINGSRC_TIM4_OC3

  On STM32G4 series, blanking sources are linked to COMP instance (except those without COMPx suffix that are common to all instances) Note: For COMPx & TIMx instances availability, please refer to datasheet

Notes

- Availability of parameters of blanking source from timer depends on timers availability on the selected device.
- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR BLANKING LL_COMP_GetOutputBlankingSource

LL_COMP_Enable

Function name

__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)

Function description

Enable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- After enable from off state, comparator requires a delay to reach reach propagation delay specification. Refer to device datasheet, parameter "tSTART".

Reference Manual to LL API cross reference:

- CSR EN LL_COMP_Enable

LL_COMP_Disable

Function name

__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)

Function description

Disable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_COMP_IsEnabled (COMP_TypeDef * COMPx)

Function description

Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EN LL_COMP_IsEnabled

LL_COMP_Lock

Function name

__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)

Function description

Lock comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CSR LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name

__STATIC_INLINE uint32_t LL_COMP_IsLocked (COMP_TypeDef * COMPx)

Function description

Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CSR LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name

__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (COMP_TypeDef * COMPx)

Function description

Read comparator instance output level.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUT_LEVEL_LOW
  - LL_COMP_OUTPUT_LEVEL_HIGH

Notes

- On this STM32 serie, comparator 'value' is taken before polarity and blanking are applied, thus: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus

Reference Manual to LL API cross reference:

- CSR VALUE LL_COMP_ReadOutputLevel

LL_COMP_DeInit

Function name

ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)

Function description

De-initialize registers of the selected COMP instance to their default reset values.

Parameters

- **COMPx:** COMP instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: COMP registers are de-initialized
  - ERROR: COMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name

ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Initialize some features of COMP instance.

Parameters

- **COMPx:** COMP instance
- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: COMP registers are initialized
  - ERROR: COMP registers are not initialized

Notes

- This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name

void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Set each LL_COMP_InitTypeDef field to default value.

Parameters

- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

67.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

67.3.1 COMP

COMP

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_STARTUP_US

Delay for COMP startup time

LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US

Delay for COMP voltage scaler stabilization time

Comparator input - Hysteresis

LL_COMP_HYSTERESIS_NONE

No hysteresis

LL_COMP_HYSTERESIS_10MV

Hysteresis level 10mV

LL_COMP_HYSTERESIS_20MV

Hysteresis level 20mV

LL_COMP_HYSTERESIS_30MV

Hysteresis level 30mV

LL_COMP_HYSTERESIS_40MV

Hysteresis level 40mV

LL_COMP_HYSTERESIS_50MV

Hysteresis level 50mV

LL_COMP_HYSTERESIS_60MV

Hysteresis level 60mV

LL_COMP_HYSTERESIS_70MV

Hysteresis level 70mV

LL_COMP_HYSTERESIS_LOW

Hysteresis level low

LL_COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

LL_COMP_HYSTERESIS_HIGH

Hysteresis level high

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

LL_COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

LL_COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

LL_COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

LL_COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 Channel 1 for COMP1/3/4. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 Channel 2 for COMP2/5. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_DAC2_CH1

Comparator input minus connected to DAC2 Channel 1 for COMP6/7. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_DAC3_CH1

Comparator input minus connected to DAC3 Channel 1 for COMP1/3. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_DAC3_CH2

Comparator input minus connected to DAC3 Channel 2 for COMP2/4. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_DAC4_CH1

Comparator input minus connected to DAC4 Channel 1 for COMP5/7. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_DAC4_CH2

Comparator input minus connected to DAC4 Channel 2 for COMP6. Note: For COMPx & DACx instances availability, please refer to datasheet

LL_COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PA4 for COMP1, pin PA5 for COMP2, pin PF1 for COMP3, pin PE8 for COMP4, pin PB10 for COMP5, pin PD10 for COMP6, pin PD15 for COMP7). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PA0 for COMP1, pin PA2 for COMP2, pin PC0 for COMP3, pin PB2 for COMP4, pin PD13 for COMP5, pin PB15 for COMP6, pin PB12 for COMP7). Note: For COMPx instance availability, please refer to datasheet

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PA1 for COMP1, pin PA7 for COMP2, pin PA0 for COMP3, pin PB0 for COMP4, pin PB13 for COMP5, pin PB11 for COMP6, pin PB14 for COMP7). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB1 for COMP1, pin PA3 for COMP2, pin PC1 for COMP3, pin PE7 for COMP4, pin PD12 for COMP5, pin PD11 for COMP6, pin PD14 for COMP7). Note: For COMPx instance availability, please refer to datasheet

Comparator output - Blanking source

LL_COMP_BLANKINGSRC_NONE

Comparator output without blanking

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP1

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP1). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP2

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP2). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP3

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP3). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP4

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP4). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP5

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP5). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP6

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP6). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM1_OC5_COMP7

Comparator output blanking source TIM1 OC5 (specific to COMP instance: COMP7). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM2_OC3_COMP1

Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP1). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM2_OC3_COMP2

Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP2). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM2_OC3_COMP5

Comparator output blanking source TIM2 OC3 (specific to COMP instance: COMP5). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM2_OC4_COMP3

Comparator output blanking source TIM2 OC4 (specific to COMP instance: COMP3). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM2_OC4_COMP6

Comparator output blanking source TIM2 OC4 (specific to COMP instance: COMP6). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM3_OC3_COMP1

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP1). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM3_OC3_COMP2

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP2). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM3_OC3_COMP3

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP3). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM3_OC3_COMP5

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP5). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM3_OC3_COMP7

Comparator output blanking source TIM3 OC3 (specific to COMP instance: COMP7). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM3_OC4_COMP4

Comparator output blanking source TIM3 OC4 (specific to COMP instance: COMP4). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP1

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP1). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP2

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP2). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP3

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP3). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP4

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP4). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP5

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP5). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP6

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP6). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM8_OC5_COMP7

Comparator output blanking source TIM8 OC5 (specific to COMP instance: COMP7). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM15_OC1_COMP4

Comparator output blanking source TIM15 OC1 (specific to COMP instance: COMP4). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM15_OC2_COMP6

Comparator output blanking source TIM15 OC2 (specific to COMP instance: COMP6). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM15_OC2_COMP7

Comparator output blanking source TIM15 OC3 (specific to COMP instance: COMP7). Note: For COMPx instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM20_OC5

Comparator output blanking source TIM20 OC5 (Common to all COMP instances). Note: For TIM20 instance availability, please refer to datasheet

LL_COMP_BLANKINGSRC_TIM15_OC1

Comparator output blanking source TIM15 OC1 (Common to all COMP instances).

LL_COMP_BLANKINGSRC_TIM4_OC3

Comparator output blanking source TIM4 OC3 (Common to all COMP instances).

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_LOW

Comparator output level low (if the polarity is not inverted, otherwise to be complemented)

LL_COMP_OUTPUT_LEVEL_HIGH

Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED

COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input

LL_COMP_OUTPUTPOL_INVERTED

COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Common write and read registers macro

LL_COMP_WriteReg

Description:

- Write a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_COMP_ReadReg

Description:

- Read a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be read

Return value:

- Register: value

68 LL CORDIC Generic Driver

68.1 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

68.1.1 Detailed description of functions

LL_CORDIC_Config

Function name

__STATIC_INLINE void LL_CORDIC_Config (CORDIC_TypeDef * CORDICx, uint32_t Function, uint32_t Precision, uint32_t Scale, uint32_t NbWrite, uint32_t NbRead, uint32_t InSize, uint32_t OutSize)

Function description

Configure the CORDIC processing.

Parameters

- **CORDICx:** CORDIC instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Notes

- This function set all parameters of CORDIC processing. These parameters can also be set individually using dedicated functions:
  LL_CORDIC_SetFunction()LL_CORDIC_SetPrecision()LL_CORDIC_SetScale()LL_CORDIC_SetNbWrite( )LL_CORDIC_SetNbRead()LL_CORDIC_SetInSize()LL_CORDIC_SetOutSize()

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_Configure
- CSR PRECISION LL_CORDIC_Configure
- CSR SCALE LL_CORDIC_Configure
- CSR NARGS LL_CORDIC_Configure
- CSR NRES LL_CORDIC_Configure
- CSR ARGSIZE LL_CORDIC_Configure
- CSR RESSIZE LL_CORDIC_Configure

LL_CORDIC_SetFunction

Function name

__STATIC_INLINE void LL_CORDIC_SetFunction (CORDIC_TypeDef * CORDICx, uint32_t Function)

Function description

Configure function.

Parameters

- **CORDICx:** CORDIC Instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_SetFunction

LL_CORDIC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetFunction (CORDIC_TypeDef * CORDICx)

Function description

Return function.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_GetFunction

LL_CORDIC_SetPrecision

Function name

__STATIC_INLINE void LL_CORDIC_SetPrecision (CORDIC_TypeDef * CORDICx, uint32_t Precision)

Function description

Configure precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_SetPrecision

LL_CORDIC_GetPrecision

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetPrecision (CORDIC_TypeDef * CORDICx)

Function description

Return precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_GetPrecision

LL_CORDIC_SetScale

Function name

__STATIC_INLINE void LL_CORDIC_SetScale (CORDIC_TypeDef * CORDICx, uint32_t Scale)

Function description

Configure scaling factor.

Parameters

- **CORDICx:** CORDIC Instance
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_SetScale

LL_CORDIC_GetScale

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetScale (CORDIC_TypeDef * CORDICx)

Function description

Return scaling factor.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_GetScale

LL_CORDIC_SetNbWrite

Function name

__STATIC_INLINE void LL_CORDIC_SetNbWrite (CORDIC_TypeDef * CORDICx, uint32_t NbWrite)

Function description

Configure number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_SetNbWrite

LL_CORDIC_GetNbWrite

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbWrite (CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_GetNbWrite

LL_CORDIC_SetNbRead

Function name

__STATIC_INLINE void LL_CORDIC_SetNbRead (CORDIC_TypeDef * CORDICx, uint32_t NbRead)

Function description

Configure number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_SetNbRead

LL_CORDIC_GetNbRead

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbRead (CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_GetNbRead

LL_CORDIC_SetInSize

Function name

__STATIC_INLINE void LL_CORDIC_SetInSize (CORDIC_TypeDef * CORDICx, uint32_t InSize)

Function description

Configure width of input data.

Parameters

- **CORDICx:** CORDIC Instance
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_SetInSize

LL_CORDIC_GetInSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetInSize (CORDIC_TypeDef * CORDICx)

Function description

Return width of input data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_GetInSize

LL_CORDIC_SetOutSize

Function name

__STATIC_INLINE void LL_CORDIC_SetOutSize (CORDIC_TypeDef * CORDICx, uint32_t OutSize)

Function description

Configure width of output data.

Parameters

- **CORDICx:** CORDIC Instance
- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RESSIZE LL_CORDIC_SetOutSize

LL_CORDIC_GetOutSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetOutSize (CORDIC_TypeDef * CORDICx)

Function description

Return width of output data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR RESSIZE LL_CORDIC_GetOutSize

LL_CORDIC_EnableIT

Function name

__STATIC_INLINE void LL_CORDIC_EnableIT (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_EnableIT

LL_CORDIC_DisableIT

Function name

__STATIC_INLINE void LL_CORDIC_DisableIT (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_DisableIT

LL_CORDIC_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledIT (CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready interrupt state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_IsEnabledIT

LL_CORDIC_EnableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_EnableDMAReq_RD

LL_CORDIC_DisableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_DisableDMAReq_RD

LL_CORDIC_IsEnabledDMAReq_RD

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA read channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_IsEnabledDMAReq_RD

LL_CORDIC_EnableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_EnableDMAReq_WR

LL_CORDIC_DisableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_DisableDMAReq_WR

LL_CORDIC_IsEnabledDMAReq_WR

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA write channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_IsEnabledDMAReq_WR

LL_CORDIC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_CORDIC_DMA_GetRegAddr (CORDIC_TypeDef * CORDICx, uint32_t Direction)

Function description

Get the CORDIC data register address used for DMA transfer.

Parameters

- **CORDICx:** CORDIC Instance
- **Direction:** parameter can be one of the following values:
  - LL_CORDIC_DMA_REG_DATA_IN
  - LL_CORDIC_DMA_REG_DATA_OUT

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_DMA_GetRegAddr
-
- WDATA ARG LL_CORDIC_DMA_GetRegAddr

LL_CORDIC_IsActiveFlag_RRDY

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsActiveFlag_RRDY (CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready flag state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR RRDY LL_CORDIC_IsActiveFlag_RRDY

LL_CORDIC_WriteData

Function name

__STATIC_INLINE void LL_CORDIC_WriteData (CORDIC_TypeDef * CORDICx, uint32_t InData)

Function description

Write 32-bit input data for the CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance
- **InData:** 0 .. 0xFFFFFFFF : 32-bit value to be provided as input data for CORDIC processing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA ARG LL_CORDIC_WriteData

LL_CORDIC_ReadData

Function name

__STATIC_INLINE uint32_t LL_CORDIC_ReadData (CORDIC_TypeDef * CORDICx)

Function description

Return 32-bit output data of CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **32-bit:** output data of CORDIC processing.

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_ReadData

LL_CORDIC_DeInit

Function name

ErrorStatus LL_CORDIC_DeInit (CORDIC_TypeDef * CORDICx)

Function description

De-Initialize CORDIC peripheral registers to their default reset values.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CORDIC registers are de-initialized
  - ERROR: CORDIC registers are not de-initialized

68.2 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

68.2.1 CORDIC

CORDIC

DMA register data

LL_CORDIC_DMA_REG_DATA_IN

Get address of input data register

LL_CORDIC_DMA_REG_DATA_OUT

Get address of output data register

FUNCTION

LL_CORDIC_FUNCTION_COSINE

Cosine

LL_CORDIC_FUNCTION_SINE

Sine

LL_CORDIC_FUNCTION_PHASE

Phase

LL_CORDIC_FUNCTION_MODULUS

Modulus

LL_CORDIC_FUNCTION_ARCTANGENT

Arctangent

LL_CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

LL_CORDIC_FUNCTION_HSINE

Hyperbolic Sine

LL_CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

LL_CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

LL_CORDIC_FUNCTION_SQUAREROOT

Square Root

Get Flags Defines

LL_CORDIC_FLAG_RRDY

INSIZE

LL_CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

LL_CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

IT Defines

LL_CORDIC_IT_IEN

Result Ready interrupt enable

NBREAD

LL_CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data ouput (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

NBWRITE

LL_CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

OUTSIZE

LL_CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

LL_CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

PRECISION

LL_CORDIC_PRECISION_1CYCLE

LL_CORDIC_PRECISION_2CYCLES

LL_CORDIC_PRECISION_3CYCLES

LL_CORDIC_PRECISION_4CYCLES

LL_CORDIC_PRECISION_5CYCLES

LL_CORDIC_PRECISION_6CYCLES

LL_CORDIC_PRECISION_7CYCLES

LL_CORDIC_PRECISION_8CYCLES

LL_CORDIC_PRECISION_9CYCLES

LL_CORDIC_PRECISION_10CYCLES

LL_CORDIC_PRECISION_11CYCLES

LL_CORDIC_PRECISION_12CYCLES

LL_CORDIC_PRECISION_13CYCLES

LL_CORDIC_PRECISION_14CYCLES

LL_CORDIC_PRECISION_15CYCLES

SCALE

LL_CORDIC_SCALE_0

LL_CORDIC_SCALE_1

LL_CORDIC_SCALE_2

LL_CORDIC_SCALE_3

LL_CORDIC_SCALE_4

LL_CORDIC_SCALE_5

LL_CORDIC_SCALE_6

LL_CORDIC_SCALE_7

Common Write and read registers Macros

LL_CORDIC_WriteReg

Description:

- Write a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CORDIC_ReadReg

Description:

- Read a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be read

Return value:

- Register: value

69 LL CORTEX Generic Driver

69.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

69.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)

Function description

This function checks if the Systick counter flag is active or not.

Return values

- **State:** of bit (1 or 0).

Notes

- It can be used in timeout function on application side.

Reference Manual to LL API cross reference:

- STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name

__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)

Function description

Configures the SysTick clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)

Function description

Get the SysTick clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name

__STATIC_INLINE void LL_SYSTICK_EnableIT (void)

Function description

Enable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name

__STATIC_INLINE void LL_SYSTICK_DisableIT (void)

Function description

Disable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)

Function description

Checks if the SYSTICK interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name

__STATIC_INLINE void LL_LPM_EnableSleep (void)

Function description

Processor uses sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name

__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)

Function description

Processor uses deep sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)

Function description

Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)

Function description

Do not sleep when returning to Thread mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)

Function description

Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)

Function description

Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name

__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)

Function description

Enable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name

__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)

Function description

Disable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)

Function description

Get Implementer code.

Return values

- **Value:** should be equal to 0x41 for ARM

Reference Manual to LL API cross reference:

- SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)

Function description

Get Variant number (The r value in the rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetArchitecture

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void)

Function description

Get Architecture number.

Return values

- **Value:** should be equal to 0xF for Cortex-M4 devices

Reference Manual to LL API cross reference:

- SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture

LL_CPUID_GetParNo

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)

Function description

Get Part number.

Return values

- **Value:** should be equal to 0xC24 for Cortex-M4

Reference Manual to LL API cross reference:

- SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)

Function description

Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)

Return values

- **Value:** between 0 and 255 (0x1: patch 1)

Reference Manual to LL API cross reference:

- SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name

__STATIC_INLINE void LL_MPU_Enable (uint32_t Options)

Function description

Enable MPU with input options.

Parameters

- **Options:** This parameter can be one of the following values:
  - LL_MPU_CTRL_HFNMI_PRIVDEF_NONE
  - LL_MPU_CTRL_HARDFAULT_NMI
  - LL_MPU_CTRL_PRIVILEGED_DEFAULT
  - LL_MPU_CTRL_HFNMI_PRIVDEF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name

__STATIC_INLINE void LL_MPU_Disable (void)

Function description

Disable MPU.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)

Function description

Check if MPU is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_EnableRegion

Function name

__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)

Function description

Enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RASR ENABLE LL_MPU_EnableRegion

LL_MPU_ConfigRegion

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t SubRegionDisable, uint32_t Address, uint32_t Attributes)

Function description

Configure and enable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Address:** Value of region base address
- **SubRegionDisable:** Sub-region disable value between Min_Data = 0x00 and Max_Data = 0xFF
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_REGION_SIZE_32B or LL_MPU_REGION_SIZE_64B or LL_MPU_REGION_SIZE_128B or LL_MPU_REGION_SIZE_256B or LL_MPU_REGION_SIZE_512B or LL_MPU_REGION_SIZE_1KB or LL_MPU_REGION_SIZE_2KB or LL_MPU_REGION_SIZE_4KB or LL_MPU_REGION_SIZE_8KB or LL_MPU_REGION_SIZE_16KB or LL_MPU_REGION_SIZE_32KB or LL_MPU_REGION_SIZE_64KB or LL_MPU_REGION_SIZE_128KB or LL_MPU_REGION_SIZE_256KB or LL_MPU_REGION_SIZE_512KB or LL_MPU_REGION_SIZE_1MB or LL_MPU_REGION_SIZE_2MB or LL_MPU_REGION_SIZE_4MB or LL_MPU_REGION_SIZE_8MB or LL_MPU_REGION_SIZE_16MB or LL_MPU_REGION_SIZE_32MB or LL_MPU_REGION_SIZE_64MB or LL_MPU_REGION_SIZE_128MB or LL_MPU_REGION_SIZE_256MB or LL_MPU_REGION_SIZE_512MB or LL_MPU_REGION_SIZE_1GB or LL_MPU_REGION_SIZE_2GB or LL_MPU_REGION_SIZE_4GB
  - LL_MPU_REGION_NO_ACCESS or LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_PRIV_RW_URO or LL_MPU_REGION_FULL_ACCESS or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_PRIV_RO_URO
  - LL_MPU_TEX_LEVEL0 or LL_MPU_TEX_LEVEL1 or LL_MPU_TEX_LEVEL2 or LL_MPU_TEX_LEVEL4
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_SHAREABLE or LL_MPU_ACCESS_NOT_SHAREABLE
  - LL_MPU_ACCESS_CACHEABLE or LL_MPU_ACCESS_NOT_CACHEABLE
  - LL_MPU_ACCESS_BUFFERABLE or LL_MPU_ACCESS_NOT_BUFFERABLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion
- MPU_RBAR REGION LL_MPU_ConfigRegion
- MPU_RBAR ADDR LL_MPU_ConfigRegion
- MPU_RASR XN LL_MPU_ConfigRegion
- MPU_RASR AP LL_MPU_ConfigRegion
- MPU_RASR S LL_MPU_ConfigRegion
- MPU_RASR C LL_MPU_ConfigRegion
- MPU_RASR B LL_MPU_ConfigRegion
- MPU_RASR SIZE LL_MPU_ConfigRegion

LL_MPU_DisableRegion

Function name

__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)

Function description

Disable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion
- MPU_RASR ENABLE LL_MPU_DisableRegion

69.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

69.2.1 CORTEX

CORTEX

MPU Bufferable Access

LL_MPU_ACCESS_BUFFERABLE

Bufferable memory attribute

LL_MPU_ACCESS_NOT_BUFFERABLE

Not Bufferable memory attribute

MPU Cacheable Access

LL_MPU_ACCESS_CACHEABLE

Cacheable memory attribute

LL_MPU_ACCESS_NOT_CACHEABLE

Not Cacheable memory attribute

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source.

LL_SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source.

MPU Control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE

Disable NMI and privileged SW access

LL_MPU_CTRL_HARDFAULT_NMI

Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers

LL_MPU_CTRL_PRIVILEGED_DEFAULT

Enable privileged software access to default memory map

LL_MPU_CTRL_HFNMI_PRIVDEF

Enable NMI and privileged SW access

Handler Fault type

LL_HANDLER_FAULT_USG

Usage fault

LL_HANDLER_FAULT_BUS

Bus fault

LL_HANDLER_FAULT_MEM

Memory management fault

MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE

Instruction fetches enabled

LL_MPU_INSTRUCTION_ACCESS_DISABLE

Instruction fetches disabled

MPU Region Number

LL_MPU_REGION_NUMBER0

REGION Number 0

LL_MPU_REGION_NUMBER1

REGION Number 1

LL_MPU_REGION_NUMBER2

REGION Number 2

LL_MPU_REGION_NUMBER3

REGION Number 3

LL_MPU_REGION_NUMBER4

REGION Number 4

LL_MPU_REGION_NUMBER5

REGION Number 5

LL_MPU_REGION_NUMBER6

REGION Number 6

LL_MPU_REGION_NUMBER7

REGION Number 7

MPU Region Privileges

LL_MPU_REGION_NO_ACCESS

No access

LL_MPU_REGION_PRIV_RW

RW privileged (privileged access only)

LL_MPU_REGION_PRIV_RW_URO

RW privileged - RO user (Write in a user program generates a fault)

LL_MPU_REGION_FULL_ACCESS

RW privileged & user (Full access)

LL_MPU_REGION_PRIV_RO

RO privileged (privileged read only)

LL_MPU_REGION_PRIV_RO_URO

RO privileged & user (read only)

MPU Region Size

LL_MPU_REGION_SIZE_32B

32B Size of the MPU protection region

LL_MPU_REGION_SIZE_64B

64B Size of the MPU protection region

LL_MPU_REGION_SIZE_128B

128B Size of the MPU protection region

LL_MPU_REGION_SIZE_256B

256B Size of the MPU protection region

LL_MPU_REGION_SIZE_512B

512B Size of the MPU protection region

LL_MPU_REGION_SIZE_1KB

1KB Size of the MPU protection region

LL_MPU_REGION_SIZE_2KB

2KB Size of the MPU protection region

LL_MPU_REGION_SIZE_4KB

4KB Size of the MPU protection region

LL_MPU_REGION_SIZE_8KB

8KB Size of the MPU protection region

LL_MPU_REGION_SIZE_16KB

16KB Size of the MPU protection region

LL_MPU_REGION_SIZE_32KB

32KB Size of the MPU protection region

LL_MPU_REGION_SIZE_64KB

64KB Size of the MPU protection region

LL_MPU_REGION_SIZE_128KB

128KB Size of the MPU protection region

LL_MPU_REGION_SIZE_256KB

256KB Size of the MPU protection region

LL_MPU_REGION_SIZE_512KB

512KB Size of the MPU protection region

LL_MPU_REGION_SIZE_1MB

1MB Size of the MPU protection region

LL_MPU_REGION_SIZE_2MB

2MB Size of the MPU protection region

LL_MPU_REGION_SIZE_4MB

4MB Size of the MPU protection region

LL_MPU_REGION_SIZE_8MB

8MB Size of the MPU protection region

LL_MPU_REGION_SIZE_16MB

16MB Size of the MPU protection region

LL_MPU_REGION_SIZE_32MB

32MB Size of the MPU protection region

LL_MPU_REGION_SIZE_64MB

64MB Size of the MPU protection region

LL_MPU_REGION_SIZE_128MB

128MB Size of the MPU protection region

LL_MPU_REGION_SIZE_256MB

256MB Size of the MPU protection region

LL_MPU_REGION_SIZE_512MB

512MB Size of the MPU protection region

LL_MPU_REGION_SIZE_1GB

1GB Size of the MPU protection region

LL_MPU_REGION_SIZE_2GB

2GB Size of the MPU protection region

LL_MPU_REGION_SIZE_4GB

4GB Size of the MPU protection region

MPU Shareable Access

LL_MPU_ACCESS_SHAREABLE

Shareable memory attribute

LL_MPU_ACCESS_NOT_SHAREABLE

Not Shareable memory attribute

MPU TEX Level

LL_MPU_TEX_LEVEL0

b000 for TEX bits

LL_MPU_TEX_LEVEL1

b001 for TEX bits

LL_MPU_TEX_LEVEL2

b010 for TEX bits

LL_MPU_TEX_LEVEL4

b100 for TEX bits

70 LL CRC Generic Driver

70.1 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

70.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name

__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)

Function description

Reset the CRC calculation unit.

Parameters

- **CRCx:** CRC Instance

Return values

- **None:**

Notes

- If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.

Reference Manual to LL API cross reference:

- CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)

Function description

Configure size of the polynomial.

Parameters

- **CRCx:** CRC Instance
- **PolySize:** This parameter can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (CRC_TypeDef * CRCx)

Function description

Return size of the polynomial.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the input data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (CRC_TypeDef * CRCx)

Function description

Return type of reversal for input data bit order.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (CRC_TypeDef * CRCx)

Function description

Configure the reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name

__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)

Function description

Initialize the Programmable initial CRC value.

Parameters

- **CRCx:** CRC Instance
- **InitCrc:** Value to be programmed in Programmable initial CRC value register

Return values

- **None:**

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to write the correct value
- LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInitialData (CRC_TypeDef * CRCx)

Function description

Return current Initial CRC value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable initial CRC value register

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to read the correct value

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_GetInitialData

LL_CRC_SetPolynomialCoef

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)

Function description

Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).

Parameters

- **CRCx:** CRC Instance
- **PolynomCoef:** Value to be programmed in Programmable Polynomial value register

Return values

- **None:**

Notes

- LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter.
- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (CRC_TypeDef * CRCx)

Function description

Return current Programmable polynomial value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable Polynomial value register

Notes

- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name

__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Write given 32-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name

__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)

Function description

Write given 16-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name

__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)

Function description

Write given 8-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name

__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (32 bits).

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name

__STATIC_INLINE uint16_t LL_CRC_ReadData16 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (16 bits).

Notes

- This function is expected to be used in a 16 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData8 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (8 bits).

Notes

- This function is expected to be used in a 8 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData7 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (7 bits).

Notes

- This function is expected to be used in a 7 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name

__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef * CRCx)

Function description

Return data stored in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** stored in CRC_IDR register (General-purpose 32-bit data register).

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name

__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Store data in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be stored in CRC_IDR register (32-bit) between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name

ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)

Function description

De-initialize CRC registers (Registers restored to their default values).

Parameters

- **CRCx:** CRC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRC registers are de-initialized
  - ERROR: CRC registers are not de-initialized

70.2 CRC Firmware driver defines

The following section lists the various define and macros of the module.

70.2.1 CRC

CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE

Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY

Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE

Input Data bit order not affected

LL_CRC_INDATA_REVERSE_BYTE

Input Data bit reversal done by byte

LL_CRC_INDATA_REVERSE_HALFWORD

Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_WORD

Input Data bit reversal done by word

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE

Output Data bit order not affected

LL_CRC_OUTDATA_REVERSE_BIT

Output Data bit reversal done by bit

Polynomial length

LL_CRC_POLYLENGTH_32B

32 bits Polynomial size

LL_CRC_POLYLENGTH_16B

16 bits Polynomial size

LL_CRC_POLYLENGTH_8B

8 bits Polynomial size

LL_CRC_POLYLENGTH_7B

7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

71 LL CRS Generic Driver

71.1 CRS Firmware driver API description

The following section lists the various functions of the CRS library.

71.1.1 Detailed description of functions

LL_CRS_EnableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)

Function description

Enable Frequency error counter.

Return values

- **None:**

Notes

- When this bit is set, the CRS_CFGR register is write-protected and cannot be modified

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_EnableFreqErrorCounter

LL_CRS_DisableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)

Function description

Disable Frequency error counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_DisableFreqErrorCounter

LL_CRS_IsEnabledFreqErrorCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)

Function description

Check if Frequency error counter is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_IsEnabledFreqErrorCounter

LL_CRS_EnableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)

Function description

Enable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

LL_CRS_DisableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void)

Function description

Disable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_DisableAutoTrimming

LL_CRS_IsEnabledAutoTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void)

Function description

Check if Automatic trimming is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming

LL_CRS_SetHSI48SmoothTrimming

Function name

__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)

Function description

Set HSI48 oscillator smooth trimming.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 63

Return values

- **None:**

Notes

- When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only
- Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_SetHSI48SmoothTrimming

LL_CRS_GetHSI48SmoothTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void)

Function description

Get HSI48 oscillator smooth trimming.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 63

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_GetHSI48SmoothTrimming

LL_CRS_SetReloadCounter

Function name

__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)

Function description

Set counter reload value.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 0xFFFF

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALC_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)

Function description

Get counter reload value.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name

__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)

Function description

Set frequency error limit.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 255

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)

Function description

Get frequency error limit.

Return values

- **A:** number between Min_Data = 0 and Max_Data = 255

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_GetFreqErrorLimit

LL_CRS_SetSyncDivider

Function name

__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)

Function description

Set division factor for SYNC signal.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)

Function description

Get division factor for SYNC signal.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name

__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)

Function description

Set SYNC signal source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)

Function description

Get SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name

__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)

Function description

Set input polarity for the SYNC signal source.

Parameters

- **Polarity:** This parameter can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)

Function description

Get input polarity for the SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name

__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)

Function description

Configure CRS for the synchronization.

Parameters

- **HSI48CalibrationValue:** a number between Min_Data = 0 and Max_Data = 63
- **ErrorLimitValue:** a number between Min_Data = 0 and Max_Data = 0xFFFF
- **ReloadValue:** a number between Min_Data = 0 and Max_Data = 255
- **Settings:** This parameter can be a combination of the following values:
  - LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128
  - LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB
  - LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_ConfigSynchronization
- CFGR RELOAD LL_CRS_ConfigSynchronization
- CFGR FELIM LL_CRS_ConfigSynchronization
- CFGR SYNCDIV LL_CRS_ConfigSynchronization
- CFGR SYNCSRC LL_CRS_ConfigSynchronization
- CFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name

__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)

Function description

Generate software SYNC event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)

Function description

Get the frequency error direction latched in the time of the last SYNC event.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_FREQ_ERROR_DIR_UP
  - LL_CRS_FREQ_ERROR_DIR_DOWN

Reference Manual to LL API cross reference:

- ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)

Function description

Get the frequency error counter value latched in the time of the last SYNC event.

Return values

- **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)

Function description

Check if SYNC event OK signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)

Function description

Check if SYNC warning signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)

Function description

Check if Synchronization or trimming error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)

Function description

Check if Expected SYNC signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)

Function description

Check if SYNC error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)

Function description

Check if SYNC missed error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)

Function description

Check if Trimming overflow or underflow occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)

Function description

Clear the SYNC event OK flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)

Function description

Clear the SYNC warning flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCWARNC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)

Function description

Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)

Function description

Clear Expected SYNC flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)

Function description

Enable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)

Function description

Disable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK (void)

Function description

Check if SYNC event OK interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)

Function description

Enable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)

Function description

Disable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)

Function description

Check if SYNC warning interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)

Function description

Enable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)

Function description

Disable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)

Function description

Check if Synchronization or trimming error interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)

Function description

Enable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)

Function description

Disable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)

Function description

Check if Expected SYNC interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name

ErrorStatus LL_CRS_DeInit (void)

Function description

De-Initializes CRS peripheral registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRS registers are de-initialized
  - ERROR: not applicable

71.2 CRS Firmware driver defines

The following section lists the various define and macros of the module.

71.2.1 CRS

CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT

Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT

LL_CRS_HSI48CALIBRATION_DEFAULT

Notes:

- The default value is 64, which corresponds to the middle of the trimming interval. The trimming step is specified in the product datasheet. A higher TRIM value corresponds to a higher output frequency

Frequency Error Direction

LL_CRS_FREQ_ERROR_DIR_UP

Upcounting direction, the actual frequency is above the target

LL_CRS_FREQ_ERROR_DIR_DOWN

Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

LL_CRS_ISR_SYNCMISS

LL_CRS_ISR_TRIMOVF

IT Defines

LL_CRS_CR_SYNCOKIE

LL_CRS_CR_SYNCWARNIE

LL_CRS_CR_ERRIE

LL_CRS_CR_ESYNCIE

Synchronization Signal Divider

LL_CRS_SYNC_DIV_1

Synchro Signal not divided (default)

LL_CRS_SYNC_DIV_2

Synchro Signal divided by 2

LL_CRS_SYNC_DIV_4

Synchro Signal divided by 4

LL_CRS_SYNC_DIV_8

Synchro Signal divided by 8

LL_CRS_SYNC_DIV_16

Synchro Signal divided by 16

LL_CRS_SYNC_DIV_32

Synchro Signal divided by 32

LL_CRS_SYNC_DIV_64

Synchro Signal divided by 64

LL_CRS_SYNC_DIV_128

Synchro Signal divided by 128

Synchronization Signal Polarity

LL_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

LL_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

Synchronization Signal Source

LL_CRS_SYNC_SOURCE_GPIO

Synchro Signal soucre GPIO

LL_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

LL_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

Exported_Macros_Calculate_Reload

__LL_CRS_CALC_CALCULATE_RELOADVALUE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

Common Write and read registers Macros

LL_CRS_WriteReg

Description:

- Write a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg

Description:

- Read a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be read

Return value:

- Register: value

72 LL DAC Generic Driver

72.1 DAC Firmware driver registers structures

72.1.1 LL_DAC_InitTypeDef

LL_DAC_InitTypeDef is defined in the stm32g4xx_ll_dac.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t TriggerSource2***
- ***uint32_t WaveAutoGeneration***
- ***uint32_t WaveAutoGenerationConfig***
- ***uint32_t OutputBuffer***
- ***uint32_t OutputConnection***
- ***uint32_t OutputMode***

Field Documentation

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource***

  Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function **LL_DAC_SetTriggerSource()**.

  Note:

  – If waveform automatic generation mode is set to sawtooth, this parameter is used as sawtooth RESET trigger

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource2***

  Set the conversion secondary trigger source for the selected DAC channel: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function LL_DAC_SetTriggerSource2().

  Note:

  – If waveform automatic generation mode is set to sawtooth, this parameter is used as sawtooth step trigger

- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration***

  Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_WAVE_AUTO_GENERATION_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetWaveAutoGeneration()**.

- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig***

  Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of ***DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS*** If waveform automatic generation mode is set to triangle, this parameter can be a value of ***DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE*** If waveform automatic generation mode is set to sawtooth, this parameter host the sawtooth configuration: polarity, reset data, increment data. Use __LL_DAC_FORMAT_SAWTOOTHWAVECONFIG macro to set this parameter value.

  Note:

  – If waveform automatic generation mode is disabled, this parameter is discarded.

  This feature can be modified afterwards using unitary function **LL_DAC_SetWaveNoiseLFSR()**, **LL_DAC_SetWaveTriangleAmplitude()**, **LL_DAC_SetWaveSawtoothPolarity()**, **LL_DAC_SetWaveSawtoothResetData()** or **LL_DAC_SetWaveSawtoothStepData()**, depending on the wave automatic generation selected.

- ***uint32_t LL_DAC_InitTypeDef::OutputBuffer***
  Set the output buffer for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_BUFFER***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputBuffer()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputConnection***
  Set the output connection for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_CONNECTION***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputConnection()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputMode***
  Set the output mode normal or sample-and-hold for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputMode()**.

72.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

72.2.1 Detailed description of functions

LL_DAC_SetHighFrequencyMode

Function name

__STATIC_INLINE void LL_DAC_SetHighFrequencyMode (DAC_TypeDef * DACx, uint32_t HighFreqMode)

Function description

Set the high frequency interface mode for the selected DAC instance.

Parameters

- **DACx:** DAC instance
- **HighFreqMode:** This parameter can be one of the following values:
  - LL_DAC_HIGH_FREQ_MODE_DISABLE
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR HFSEL LL_DAC_SetHighFrequencyMode

LL_DAC_GetHighFrequencyMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetHighFrequencyMode (DAC_TypeDef * DACx)

Function description

Get the high frequency interface mode for the selected DAC instance.

Parameters

- **DACx:** DAC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_HIGH_FREQ_MODE_DISABLE
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

Reference Manual to LL API cross reference:

- MCR HFSEL LL_DAC_GetHighFrequencyMode

LL_DAC_SetMode

Function name

__STATIC_INLINE void LL_DAC_SetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t ChannelMode)

Function description

Set the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **ChannelMode:** This parameter can be one of the following values:
  - LL_DAC_MODE_NORMAL_OPERATION
  - LL_DAC_MODE_CALIBRATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_SetMode
- CR CEN2 LL_DAC_SetMode

LL_DAC_GetMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_MODE_NORMAL_OPERATION
  - LL_DAC_MODE_CALIBRATION

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_GetMode
- CR CEN2 LL_DAC_GetMode

LL_DAC_SetTrimmingValue

Function name

__STATIC_INLINE void LL_DAC_SetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TrimmingValue)

Function description

Set the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_SetTrimmingValue
- CCR OTRIM2 LL_DAC_SetTrimmingValue

LL_DAC_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_GetTrimmingValue
- CCR OTRIM2 LL_DAC_GetTrimmingValue

LL_DAC_SetTriggerSource

Function name

__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)

Function description

Set the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **TriggerSource:** This parameter can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE9
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG1 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG2 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG3 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG4 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG5 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG6 (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO1 (3) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO2 (4) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO3 (1) (5)

  (1) On this STM32 serie, parameter only available on DAC3. (2) On this STM32 serie, parameter only available on DAC1/2/4. (3) On this STM32 serie, parameter only available on DAC1&4. (4) On this STM32 serie, parameter only available on DAC2. Refer to device datasheet for DACx instances availability. (5) On this STM32 serie, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list)

Return values

- **None:**

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_SetTriggerSource
- CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE9
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG1 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG2 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG3 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG4 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG5 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG6 (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO1 (3) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO2 (4) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO3 (1) (5)

  (1) On this STM32 serie, parameter only available on DAC3. (2) On this STM32 serie, parameter only available on DAC1/2/4. (3) On this STM32 serie, parameter only available on DAC1&4. (4) On this STM32 serie, parameter only available on DAC2. Refer to device datasheet for DACx instances availability. (5) On this STM32 serie, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list)

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_GetTriggerSource
- CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name

__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)

Function description

Set the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **WaveAutoGeneration:** This parameter can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
  - LL_DAC_WAVE_AUTO_GENERATION_SAWTOOTH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_SetWaveAutoGeneration
- CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE
  - LL_DAC_WAVE_AUTO_GENERATION_SAWTOOTH

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_GetWaveAutoGeneration
- CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name

__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)

Function description

Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **NoiseLFSRMask:** This parameter can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveNoiseLFSR
- CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name

__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)

Function description

Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_SetWaveSawtoothPolarity

Function name

__STATIC_INLINE void LL_DAC_SetWaveSawtoothPolarity (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Polarity)

Function description

Set the swatooth waveform generation polarity.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Polarity:** This parameter can be one of the following values:
  - LL_DAC_SAWTOOTH_POLARITY_DECREMENT
  - LL_DAC_SAWTOOTH_POLARITY_INCREMENT

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- STR1 STDIR1 LL_DAC_SetWaveSawtoothPolarity
- STR2 STDIR2 LL_DAC_SetWaveSawtoothPolarity

LL_DAC_GetWaveSawtoothPolarity

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveSawtoothPolarity (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sawtooth waveform generation polarity.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_SAWTOOTH_POLARITY_DECREMENT
  - LL_DAC_SAWTOOTH_POLARITY_INCREMENT

Reference Manual to LL API cross reference:

- STR1 STDIR1 LL_DAC_GetWaveSawtoothPolarity
- STR2 STDIR2 LL_DAC_GetWaveSawtoothPolarity

LL_DAC_SetWaveSawtoothResetData

Function name

__STATIC_INLINE void LL_DAC_SetWaveSawtoothResetData (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t ResetData)

Function description

Set the swatooth waveform generation reset data.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **ResetData:** This parameter is the sawtooth reset value. Range is from 0 to DAC full range 4095 (0xFFF)

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- STR1 STRSTDATA1 LL_DAC_SetWaveSawtoothResetData
- STR2 STRSTDATA2 LL_DAC_SetWaveSawtoothResetData

LL_DAC_GetWaveSawtoothResetData

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveSawtoothResetData (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sawtooth waveform generation reset data.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value is the sawtooth reset value. Range is from 0 to DAC full range 4095 (0xFFF)

Reference Manual to LL API cross reference:

- STR1 STRSTDATA1 LL_DAC_GetWaveSawtoothResetData
- STR2 STRSTDATA2 LL_DAC_GetWaveSawtoothResetData

LL_DAC_SetWaveSawtoothStepData

Function name

__STATIC_INLINE void LL_DAC_SetWaveSawtoothStepData (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t StepData)

Function description

Set the swatooth waveform generation step data.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **StepData:** This parameter is the sawtooth step value. 12.4 bit format, unsigned: 12 bits exponent / 4 bits mantissa Step value step is 1/16 = 0.0625 Step value range is 0.0000 to 4095.9375 (0xFFF.F)

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- STR1 STINCDATA1 LL_DAC_SetWaveSawtoothStepData
- STR2 STINCDATA2 LL_DAC_SetWaveSawtoothStepData

LL_DAC_GetWaveSawtoothStepData

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveSawtoothStepData (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sawtooth waveform generation step data.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value is the sawtooth step value. 12.4 bit format, unsigned: 12 bits exponent / 4 bits mantissa Step value step is 1/16 = 0.0625 Step value range is 0.0000 to 4095.9375 (0xFFF.F)

Reference Manual to LL API cross reference:

- STR1 STINCDATA1 LL_DAC_GetWaveSawtoothStepData
- STR2 STINCDATA2 LL_DAC_GetWaveSawtoothStepData

LL_DAC_SetWaveSawtoothResetTriggerSource

Function name

__STATIC_INLINE void LL_DAC_SetWaveSawtoothResetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)

Function description

Set the swatooth waveform generation reset trigger source.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **TriggerSource:** This parameter can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE9
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG1 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG2 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG3 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG4 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG5 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG6 (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO1 (3) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO2 (4) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO3 (1) (5)

  (1) On this STM32 serie, parameter only available on DAC3. (2) On this STM32 serie, parameter only available on DAC1/2/4. (3) On this STM32 serie, parameter only available on DAC1&4. (4) On this STM32 serie, parameter only available on DAC2. Refer to device datasheet for DACx instances availability. (5) On this STM32 serie, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list)

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- STMODR STRSTTRIGSEL1 LL_DAC_SetWaveSawtoothResetTriggerSource
- STMODR STRSTTRIGSEL2 LL_DAC_SetWaveSawtoothResetTriggerSource

LL_DAC_GetWaveSawtoothResetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveSawtoothResetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sawtooth waveform generation reset trigger source.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE9
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG1 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG2 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG3 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG4 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG5 (5)
  - LL_DAC_TRIG_EXT_HRTIM_RST_TRG6 (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO1 (3) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO2 (4) (5)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO3 (1) (5)

  (1) On this STM32 serie, parameter only available on DAC3. (2) On this STM32 serie, parameter only available on DAC1/2/4. (3) On this STM32 serie, parameter only available on DAC1&4. (4) On this STM32 serie, parameter only available on DAC2. Refer to device datasheet for DACx instances availability. (5) On this STM32 serie, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list)

Reference Manual to LL API cross reference:

- STMODR STRSTTRIGSEL1 LL_DAC_GetWaveSawtoothResetTriggerSource
- STMODR STRSTTRIGSEL2 LL_DAC_GetWaveSawtoothResetTriggerSource

LL_DAC_SetWaveSawtoothStepTriggerSource

Function name

__STATIC_INLINE void LL_DAC_SetWaveSawtoothStepTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)

Function description

Set the swatooth waveform generation step trigger source.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **TriggerSource:** This parameter can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE10
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG1 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG2 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG3 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG4 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG5 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG6 (3)

  (1) On this STM32 serie, parameter only available on DAC3. (2) On this STM32 serie, parameter only available on DAC1/2/4. Refer to device datasheet for DACx instances availability. (3) On this STM32 serie, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list)

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- STMODR STINCTRIGSEL1 LL_DAC_SetWaveSawtoothStepTriggerSource
- STMODR STINCTRIGSEL2 LL_DAC_SetWaveSawtoothStepTriggerSource

LL_DAC_GetWaveSawtoothStepTriggerSource

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveSawtoothStepTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sawtooth waveform generation step trigger source.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_EXTI_LINE10
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG1 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG2 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG3 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG4 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG5 (3)
  - LL_DAC_TRIG_EXT_HRTIM_STEP_TRG6 (3)

  (1) On this STM32 serie, parameter only available on DAC3. (2) On this STM32 serie, parameter only available on DAC1/2/4. Refer to device datasheet for DACx instances availability. (3) On this STM32 serie, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list)

Reference Manual to LL API cross reference:

- STMODR STINCTRIGSEL1 LL_DAC_GetWaveSawtoothStepTriggerSource
- STMODR STINCTRIGSEL2 LL_DAC_GetWaveSawtoothStepTriggerSource

LL_DAC_ConfigOutput

Function name

__STATIC_INLINE void LL_DAC_ConfigOutput (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode, uint32_t OutputBuffer, uint32_t OutputConnection)

Function description

Set the output for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- This function set several features: mode normal or sample-and-holdbufferconnection to GPIO or internal path. These features can also be set individually using dedicated functions:LL_DAC_SetOutputBuffer()LL_DAC_SetOutputMode()LL_DAC_SetOutputConnection()
- On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).
- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_ConfigOutput
- CR MODE2 LL_DAC_ConfigOutput

LL_DAC_SetOutputMode

Function name

__STATIC_INLINE void LL_DAC_SetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode)

Function description

Set the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Return values

- **None:**

Notes

- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputMode
- CR MODE2 LL_DAC_SetOutputMode

LL_DAC_GetOutputMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputMode
- CR MODE2 LL_DAC_GetOutputMode

LL_DAC_SetOutputBuffer

Function name

__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)

Function description

Set the output buffer for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Return values

- **None:**

Notes

- On this STM32 serie, when buffer is enabled, its offset can be trimmed: factory calibration default values can be replaced by user trimming values, using function LL_DAC_SetTrimmingValue().

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputBuffer
- CR MODE2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output buffer state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputBuffer
- CR MODE2 LL_DAC_GetOutputBuffer

LL_DAC_SetOutputConnection

Function name

__STATIC_INLINE void LL_DAC_SetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputConnection)

Function description

Set the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputConnection
- CR MODE2 LL_DAC_SetOutputConnection

LL_DAC_GetOutputConnection

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Notes

- On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputConnection
- CR MODE2 LL_DAC_GetOutputConnection

LL_DAC_SetSampleAndHoldSampleTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SampleTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **SampleTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Notes

- Sample time must be set when DAC channel is disabled or during DAC operation when DAC channel flag BWSTx is reset, otherwise the setting is ignored. Check BWSTx flag state using function "LL_DAC_IsActiveFlag_BWSTx()".

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_SetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_SetSampleAndHoldSampleTime

LL_DAC_GetSampleAndHoldSampleTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_GetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_GetSampleAndHoldSampleTime

LL_DAC_SetSampleAndHoldHoldTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t HoldTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **HoldTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_SetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_SetSampleAndHoldHoldTime

LL_DAC_GetSampleAndHoldHoldTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_GetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_GetSampleAndHoldHoldTime

LL_DAC_SetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t RefreshTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **RefreshTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_SetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_SetSampleAndHoldRefreshTime

LL_DAC_GetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_GetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_GetSampleAndHoldRefreshTime

LL_DAC_SetSignedFormat

Function name

__STATIC_INLINE void LL_DAC_SetSignedFormat (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SignedFormat)

Function description

Set the signed format for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **SignedFormat:** This parameter can be one of the following values:
  - LL_DAC_SIGNED_FORMAT_ENABLE
  - LL_DAC_SIGNED_FORMAT_DISABLE

Return values

- **None:**

Notes

- On this STM32 serie, signed format can be used to inject Q1.15, Q1.11, Q1.7 signed format data to DAC. Ex when using 12bits data format (Q1.11 is used): 0x800 will output 0v level 0xFFF will output mid-scale level 0x000 will output mid-scale level 0x7FF will output full-scale level

Reference Manual to LL API cross reference:

- MCR SINFORMAT1 LL_DAC_SetSignedFormat
- MCR SINFORMAT2 LL_DAC_SetSignedFormat

LL_DAC_GetSignedFormat

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSignedFormat (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the signed format state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_SIGNED_FORMAT_ENABLE
  - LL_DAC_SIGNED_FORMAT_DISABLE

Reference Manual to LL API cross reference:

- MCR SINFORMAT1 LL_DAC_GetSignedFormat
- MCR SINFORMAT2 LL_DAC_GetSignedFormat

LL_DAC_EnableDMAReq

Function name

__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_EnableDMAReq
- CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name

__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_DisableDMAReq
- CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA transfer request state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_IsDMAReqEnabled
- CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_EnableDMADoubleDataMode

Function name

__STATIC_INLINE void LL_DAC_EnableDMADoubleDataMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA Double data mode of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_EnableDMADoubleDataMode
- MCR DMADOUBLE2 LL_DAC_EnableDMADoubleDataMode

LL_DAC_DisableDMADoubleDataMode

Function name

__STATIC_INLINE void LL_DAC_DisableDMADoubleDataMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA Double data mode of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_DisableDMADoubleDataMode
- MCR DMADOUBLE2 LL_DAC_DisableDMADoubleDataMode

LL_DAC_IsDMADoubleDataModeEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMADoubleDataModeEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA double data mode state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_IsDMADoubleDataModeEnabled
- MCR DMADOUBLE2 LL_DAC_IsDMADoubleDataModeEnabled

LL_DAC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)

Function description

Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Register:** This parameter can be one of the following values:
  - LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  - LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  - LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values

- **DAC:** register address

Notes

- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name

__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Notes

- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name

__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Disable
- CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC enable state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_IsEnabled
- CR EN2 LL_DAC_IsEnabled

LL_DAC_IsReady

Function name

__STATIC_INLINE uint32_t LL_DAC_IsReady (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC ready for conversion state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC1RDY LL_DAC_IsReady
- SR DAC2RDY LL_DAC_IsReady

LL_DAC_EnableTrigger

Function name

__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Notes

- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_EnableTrigger
- CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name

__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_DisableTrigger
- CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC trigger state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_IsTriggerEnabled
- CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name

__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Trig DAC conversion by software for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can a combination of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Notes

- Preliminarily, DAC trigger must be set to software trigger using function LL_DAC_Init() LL_DAC_SetTriggerSource() LL_DAC_SetWaveSawtoothResetTriggerSource() (1) with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)

Reference Manual to LL API cross reference:

- SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion
- SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion

LL_DAC_TrigSWConversion2

Function name

__STATIC_INLINE void LL_DAC_TrigSWConversion2 (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Trig DAC conversion by secondary software trigger for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can a combination of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **None:**

Notes

- Preliminarily, DAC secondary trigger must be set to software trigger using function LL_DAC_Init() LL_DAC_SetWaveSawtoothStepTriggerSource() (1) with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)

Reference Manual to LL API cross reference:

- SWTRIGR SWTRIGB1 LL_DAC_TrigSWConversion2
- SWTRIGR SWTRIGB2 LL_DAC_TrigSWConversion2

LL_DAC_ConvertData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned
- DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned
- DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned
- DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned
- DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned
- DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF
- **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned
- DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name

__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Retrieve output data currently generated for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).

Reference Manual to LL API cross reference:

- DOR1 DACC1DOR LL_DAC_RetrieveOutputData
- DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_CAL1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL1 (DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG1 LL_DAC_IsActiveFlag_CAL1

LL_DAC_IsActiveFlag_CAL2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL2 (DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG2 LL_DAC_IsActiveFlag_CAL2

LL_DAC_IsActiveFlag_BWST1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST1 (DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST1 LL_DAC_IsActiveFlag_BWST1

LL_DAC_IsActiveFlag_BWST2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST2 (DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST2 LL_DAC_IsActiveFlag_BWST2

LL_DAC_IsActiveFlag_DAC1RDY

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DAC1RDY (DAC_TypeDef * DACx)

Function description

Get DAC ready status flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC1RDY LL_DAC_IsActiveFlag_DAC1RDY

LL_DAC_IsActiveFlag_DAC2RDY

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DAC2RDY (DAC_TypeDef * DACx)

Function description

Get DAC ready status flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC2RDY LL_DAC_IsActiveFlag_DAC2RDY

LL_DAC_IsActiveFlag_DORSTAT1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DORSTAT1 (DAC_TypeDef * DACx)

Function description

Get DAC output register status flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DORSTAT1 LL_DAC_IsActiveFlag_DORSTAT1

LL_DAC_IsActiveFlag_DORSTAT2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DORSTAT2 (DAC_TypeDef * DACx)

Function description

Get DAC output register status flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DORSTAT2 LL_DAC_IsActiveFlag_DORSTAT2

LL_DAC_IsActiveFlag_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2

LL_DAC_ClearFlag_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1

LL_DAC_ClearFlag_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

LL_DAC_EnableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

LL_DAC_EnableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

LL_DAC_DisableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

LL_DAC_DisableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

LL_DAC_IsEnabledIT_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

LL_DAC_IsEnabledIT_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

LL_DAC_DeInit

Function name

ErrorStatus LL_DAC_DeInit (DAC_TypeDef * DACx)

Function description

De-initialize registers of the selected DAC instance to their default reset values.

Parameters

- **DACx:** DAC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DAC registers are de-initialized
  - ERROR: not applicable

LL_DAC_Init

Function name

ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Initialize some features of DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all instances. Refer to device datasheet for channels availability.
- **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DAC registers are initialized
  - ERROR: DAC registers are not initialized

Notes

- LL_DAC_Init() aims to ease basic configuration of a DAC channel. Leaving it ready to be enabled and output: a level by calling one of LL_DAC_ConvertData12RightAligned LL_DAC_ConvertData12LeftAligned LL_DAC_ConvertData8RightAligned or one of the supported autogenerated wave.
- This function allows configuration of: Output modeTriggerWave generation
- The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC channel must be disabled.

LL_DAC_StructInit

Function name

void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Set each LL_DAC_InitTypeDef field to default value.

Parameters

- **DAC_InitStruct:** pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

72.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

72.3.1 DAC

DAC

DAC channels

LL_DAC_CHANNEL_1

DAC channel 1

LL_DAC_CHANNEL_2

DAC channel 2

DAC flags

LL_DAC_FLAG_DMAUDR1

DAC channel 1 flag DMA underrun

LL_DAC_FLAG_CAL1

DAC channel 1 flag offset calibration status

LL_DAC_FLAG_BWST1

DAC channel 1 flag busy writing sample time

LL_DAC_FLAG_DAC1RDY

DAC channel 1 flag ready

LL_DAC_FLAG_DORSTAT1

DAC channel 1 flag output register

LL_DAC_FLAG_DMAUDR2

DAC channel 2 flag DMA underrun

LL_DAC_FLAG_CAL2

DAC channel 2 flag offset calibration status

LL_DAC_FLAG_BWST2

DAC channel 2 flag busy writing sample time

LL_DAC_FLAG_DAC2RDY

DAC channel 2 flag ready

LL_DAC_FLAG_DORSTAT2

DAC channel 2 flag output register

DAC high frequency interface mode

LL_DAC_HIGH_FREQ_MODE_DISABLE

High frequency interface mode disabled

LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ

High frequency interface mode compatible to AHB>80MHz enabled

LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

High frequency interface mode compatible to AHB>160MHz enabled

Definitions of DAC hardware constraints delays

LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US

Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)

LL_DAC_DELAY_VOLTAGE_SETTLING_US

Delay for DAC channel voltage settling time

DAC interruptions

LL_DAC_IT_DMAUDRIE1

DAC channel 1 interruption DMA underrun

LL_DAC_IT_DMAUDRIE2

DAC channel 2 interruption DMA underrun

DAC operating mode

LL_DAC_MODE_NORMAL_OPERATION

DAC channel in mode normal operation

LL_DAC_MODE_CALIBRATION

DAC channel in mode calibration

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE

The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption

LL_DAC_OUTPUT_BUFFER_DISABLE

The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC channel output connection

LL_DAC_OUTPUT_CONNECT_GPIO

The selected DAC channel output is connected to external pin

LL_DAC_OUTPUT_CONNECT_INTERNAL

The selected DAC channel output is connected to on-chip peripherals via internal paths. On this STM32 serie, output connection depends on output mode (normal or sample and hold) and output buffer state. Refer to comments of function

DAC channel output mode

LL_DAC_OUTPUT_MODE_NORMAL

The selected DAC channel output is on mode normal.

LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

The selected DAC channel output is on mode sample-and-hold. Mode sample-and-hold requires an external capacitor, refer to description of function

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED

DAC channel data holding register 12 bits right aligned

LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED

DAC channel data holding register 12 bits left aligned

LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B

DAC channel resolution 12 bits

LL_DAC_RESOLUTION_8B

DAC channel resolution 8 bits

DAC wave generation - Sawtooth polarity mode

LL_DAC_SAWTOOTH_POLARITY_DECREMENT

Sawtooth wave generation, polarity is decrement

LL_DAC_SAWTOOTH_POLARITY_INCREMENT

Sawtooth wave generation, polarity is increment

DAC channel signed format

LL_DAC_SIGNED_FORMAT_DISABLE

The selected DAC channel data format is not signed

LL_DAC_SIGNED_FORMAT_ENABLE

The selected DAC channel data format is signed

DAC trigger source

LL_DAC_TRIG_SOFTWARE

DAC (all) channel conversion trigger internal (SW start)

LL_DAC_TRIG_EXT_TIM1_TRGO

DAC3 channel conversion trigger from external peripheral: TIM1 TRGO.

LL_DAC_TRIG_EXT_TIM8_TRGO

DAC1/2/4 channel conversion trigger from external peripheral: TIM8 TRGO. Refer to device datasheet for DACx instance availability.

LL_DAC_TRIG_EXT_TIM7_TRGO

DAC (all) channel conversion trigger from external peripheral: TIM7 TRGO.

LL_DAC_TRIG_EXT_TIM15_TRGO

DAC (all) channel conversion trigger from external peripheral: TIM15 TRGO.

LL_DAC_TRIG_EXT_TIM2_TRGO

DAC (all) channel conversion trigger from external peripheral: TIM2 TRGO.

LL_DAC_TRIG_EXT_TIM4_TRGO

DAC (all) channel conversion trigger from external peripheral: TIM4 TRGO.

LL_DAC_TRIG_EXT_EXTI_LINE9

DAC (all) channel conversion trigger from external peripheral: external interrupt line 9. Note: only to be used as update or reset (sawtooth generation) trigger

LL_DAC_TRIG_EXT_EXTI_LINE10

DAC (all) channel conversion trigger from external peripheral: external interrupt line 10. Note: only to be used as increment (sawtooth generation) trigger

LL_DAC_TRIG_EXT_TIM6_TRGO

DAC (all) channel conversion trigger from external peripheral: TIM6 TRGO.

LL_DAC_TRIG_EXT_TIM3_TRGO

DAC (all) channel conversion trigger from external peripheral: TIM3 TRGO.

LL_DAC_TRIG_EXT_HRTIM_STEP_TRG1

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC STEP TRIG1 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_RST_TRG1

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC RESET TRIG1 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_STEP_TRG2

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC STEP TRIG2 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_RST_TRG2

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC RESET TRIG2 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_STEP_TRG3

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC STEP TRIG3 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_RST_TRG3

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC RESET TRIG3 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_STEP_TRG4

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC STEP TRIG4 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_RST_TRG4

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC RESET TRIG4 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_STEP_TRG5

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC STEP TRIG5 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_RST_TRG5

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC RESET TRIG5 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_STEP_TRG6

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC STEP TRIG6 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_RST_TRG6

DAC (all) channel conversion trigger from external peripheral: HRTIM DAC RESET TRIG6 (only available for sawtooth wave generation). On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_TRGO1

DAC1&4 channel conversion trigger from external peripheral: HRTIM1 DACTRG1. Note: only to be used as update or reset (sawtooth generation) trigger. Refer to device datasheet for DACx instance availability. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

LL_DAC_TRIG_EXT_HRTIM_TRGO2

DAC2 channel conversion trigger from external peripheral: HRTIM1 DACTRG2. Note: only to be used as update or reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported and DAC2 instance present (refer to device datasheet for supported features list and DAC2 instance availability)

LL_DAC_TRIG_EXT_HRTIM_TRGO3

DAC3 channel conversion trigger from external peripheral: HRTIM1 DACTRG3. Note: only to be used as update or reset (sawtooth generation) trigger. On this STM32 serie, parameter only available if HRTIM feature is supported (refer to device datasheet for supported features list)

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE

DAC channel wave auto generation mode disabled.

LL_DAC_WAVE_AUTO_GENERATION_NOISE

DAC channel wave auto generation mode enabled, set generated noise waveform.

LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

DAC channel wave auto generation mode enabled, set generated triangle waveform.

LL_DAC_WAVE_AUTO_GENERATION_SAWTOOTH

DAC channel wave auto generation mode enabled, set generated sawtooth waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0

Noise wave generation, unmask LFSR bit0, for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS1_0

Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS2_0

Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS3_0

Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS4_0

Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS5_0

Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS6_0

Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS7_0

Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS8_0

Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS9_0

Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS10_0

Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1

Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_3

Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_7

Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_15

Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_31

Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_63

Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_127

Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_255

Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_511

Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_1023

Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_2047

Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_4095

Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Return value:

- 1...2

Notes:

- The input can be a value from functions where a channel number is returned.

__LL_DAC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: 1...2

Return value:

- Returned: value can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

__LL_DAC_DIGITAL_SCALE

Description:

- Helper macro to define the DAC conversion data full-scale digital value corresponding to the selected DAC resolution.

Parameters:

- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- DAC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref + and Vref- (refer to reference manual).

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

- Helper macro to calculate the DAC conversion data (unit: digital value) corresponding to a voltage (unit: mVolt).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_DAC_FORMAT_SAWTOOTHWAVECONFIG

Description:

- Helper macro to format sawtooth wave generation configuration value to be filled into WaveAutoGenerationConfig parameter of

Parameters:

- __POLARITY__: sawtooth wave polarity (must be value of
- __RESET_DATA__: sawtooth reset data.
- __STEP_DATA__: sawtooth step data

Return value:

- Sawtooth: configuration organized in DAC_STRx compatible format.

Notes:

- This helper will format information to fit in DAC_STRx register.

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

73 LL DMAMUX Generic Driver

73.1 DMAMUX Firmware driver API description

The following section lists the various functions of the DMAMUX library.

73.1.1 Detailed description of functions

LL_DMAMUX_SetRequestID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Request)

Function description

Set DMAMUX request ID for DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_DAC1_CH1
  - LL_DMAMUX_REQ_DAC1_CH2
  - LL_DMAMUX_REQ_TIM6_UP
  - LL_DMAMUX_REQ_TIM7_UP
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX
  - LL_DMAMUX_REQ_SPI2_TX
  - LL_DMAMUX_REQ_SPI3_RX
  - LL_DMAMUX_REQ_SPI3_TX
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX
  - LL_DMAMUX_REQ_I2C2_TX
  - LL_DMAMUX_REQ_I2C3_RX
  - LL_DMAMUX_REQ_I2C3_TX (*)
  - LL_DMAMUX_REQ_I2C4_RX (*)
  - LL_DMAMUX_REQ_I2C4_TX
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX
  - LL_DMAMUX_REQ_USART3_TX
  - LL_DMAMUX_REQ_UART4_RX
  - LL_DMAMUX_REQ_UART4_TX
  - LL_DMAMUX_REQ_UART5_RX (*)
  - LL_DMAMUX_REQ_UART5_TX (*)
  - LL_DMAMUX_REQ_LPUART1_RX
  - LL_DMAMUX_REQ_LPUART1_TX
  - LL_DMAMUX_REQ_ADC2
  - LL_DMAMUX_REQ_ADC3 (*)
  - LL_DMAMUX_REQ_ADC4 (*)
  - LL_DMAMUX_REQ_ADC5 (*)
  - LL_DMAMUX_REQ_QSPI (*)
  - LL_DMAMUX_REQ_DAC2_CH1 (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM1_TRIG
  - LL_DMAMUX_REQ_TIM1_COM
  - LL_DMAMUX_REQ_TIM8_CH1
  - LL_DMAMUX_REQ_TIM8_CH2

Return values

- **None:**

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 channel 1 to 8. DMAMUX channel 8 to 15 are mapped to DMA2 channel 1 to 8.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_SetRequestID

LL_DMAMUX_GetRequestID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get DMAMUX request ID for DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15 (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_DAC1_CH1
  - LL_DMAMUX_REQ_DAC1_CH2
  - LL_DMAMUX_REQ_TIM6_UP
  - LL_DMAMUX_REQ_TIM7_UP
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX
  - LL_DMAMUX_REQ_SPI2_TX
  - LL_DMAMUX_REQ_SPI3_RX
  - LL_DMAMUX_REQ_SPI3_TX
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX
  - LL_DMAMUX_REQ_I2C2_TX
  - LL_DMAMUX_REQ_I2C3_RX
  - LL_DMAMUX_REQ_I2C3_TX (*)
  - LL_DMAMUX_REQ_I2C4_RX (*)
  - LL_DMAMUX_REQ_I2C4_TX
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX
  - LL_DMAMUX_REQ_USART3_TX
  - LL_DMAMUX_REQ_UART4_RX
  - LL_DMAMUX_REQ_UART4_TX
  - LL_DMAMUX_REQ_UART5_RX (*)
  - LL_DMAMUX_REQ_UART5_TX (*)
  - LL_DMAMUX_REQ_LPUART1_RX
  - LL_DMAMUX_REQ_LPUART1_TX
  - LL_DMAMUX_REQ_ADC2
  - LL_DMAMUX_REQ_ADC3 (*)
  - LL_DMAMUX_REQ_ADC4 (*)
  - LL_DMAMUX_REQ_ADC5 (*)
  - LL_DMAMUX_REQ_QSPI (*)
  - LL_DMAMUX_REQ_DAC2_CH1 (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM1_TRIG
  - LL_DMAMUX_REQ_TIM1_COM

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 channel 1 to 8. DMAMUX channel 8 to 15 are mapped to DMA2 channel 1 to 8.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_GetRequestID

LL_DMAMUX_SetSyncRequestNb

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t RequestNb)

Function description

Set the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15
- **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR NBREQ LL_DMAMUX_SetSyncRequestNb

LL_DMAMUX_GetSyncRequestNb

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Between:** Min_Data = 1 and Max_Data = 32

Reference Manual to LL API cross reference:

- CxCR NBREQ LL_DMAMUX_GetSyncRequestNb

LL_DMAMUX_SetSyncPolarity

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of the signal on which the DMA request is synchronized.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15
- **Polarity:** This parameter can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SPOL LL_DMAMUX_SetSyncPolarity

LL_DMAMUX_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get the polarity of the signal on which the DMA request is synchronized.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Reference Manual to LL API cross reference:

- CxCR SPOL LL_DMAMUX_GetSyncPolarity

LL_DMAMUX_EnableEventGeneration

Function name

__STATIC_INLINE void LL_DMAMUX_EnableEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the Event Generation on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_EnableEventGeneration

LL_DMAMUX_DisableEventGeneration

Function name

__STATIC_INLINE void LL_DMAMUX_DisableEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the Event Generation on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_DisableEventGeneration

LL_DMAMUX_IsEnabledEventGeneration

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the Event Generation on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_IsEnabledEventGeneration

LL_DMAMUX_EnableSync

Function name

__STATIC_INLINE void LL_DMAMUX_EnableSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the synchronization mode.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_EnableSync

LL_DMAMUX_DisableSync

Function name

__STATIC_INLINE void LL_DMAMUX_DisableSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the synchronization mode.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_DisableSync

LL_DMAMUX_IsEnabledSync

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the synchronization mode is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_IsEnabledSync

LL_DMAMUX_SetSyncID

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t SyncID)

Function description

Set DMAMUX synchronization ID on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15
- **SyncID:** This parameter can be one of the following values:
  - LL_DMAMUX_SYNC_EXTI_LINE0
  - LL_DMAMUX_SYNC_EXTI_LINE1
  - LL_DMAMUX_SYNC_EXTI_LINE2
  - LL_DMAMUX_SYNC_EXTI_LINE3
  - LL_DMAMUX_SYNC_EXTI_LINE4
  - LL_DMAMUX_SYNC_EXTI_LINE5
  - LL_DMAMUX_SYNC_EXTI_LINE6
  - LL_DMAMUX_SYNC_EXTI_LINE7
  - LL_DMAMUX_SYNC_EXTI_LINE8
  - LL_DMAMUX_SYNC_EXTI_LINE9
  - LL_DMAMUX_SYNC_EXTI_LINE10
  - LL_DMAMUX_SYNC_EXTI_LINE11
  - LL_DMAMUX_SYNC_EXTI_LINE12
  - LL_DMAMUX_SYNC_EXTI_LINE13
  - LL_DMAMUX_SYNC_EXTI_LINE14
  - LL_DMAMUX_SYNC_EXTI_LINE15
  - LL_DMAMUX_SYNC_DMAMUX_CH0
  - LL_DMAMUX_SYNC_DMAMUX_CH1
  - LL_DMAMUX_SYNC_DMAMUX_CH2
  - LL_DMAMUX_SYNC_DMAMUX_CH3
  - LL_DMAMUX_SYNC_LPTIM1_OUT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SYNC_ID LL_DMAMUX_SetSyncID

LL_DMAMUX_GetSyncID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get DMAMUX synchronization ID on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_SYNC_EXTI_LINE0
  - LL_DMAMUX_SYNC_EXTI_LINE1
  - LL_DMAMUX_SYNC_EXTI_LINE2
  - LL_DMAMUX_SYNC_EXTI_LINE3
  - LL_DMAMUX_SYNC_EXTI_LINE4
  - LL_DMAMUX_SYNC_EXTI_LINE5
  - LL_DMAMUX_SYNC_EXTI_LINE6
  - LL_DMAMUX_SYNC_EXTI_LINE7
  - LL_DMAMUX_SYNC_EXTI_LINE8
  - LL_DMAMUX_SYNC_EXTI_LINE9
  - LL_DMAMUX_SYNC_EXTI_LINE10
  - LL_DMAMUX_SYNC_EXTI_LINE11
  - LL_DMAMUX_SYNC_EXTI_LINE12
  - LL_DMAMUX_SYNC_EXTI_LINE13
  - LL_DMAMUX_SYNC_EXTI_LINE14
  - LL_DMAMUX_SYNC_EXTI_LINE15
  - LL_DMAMUX_SYNC_DMAMUX_CH0
  - LL_DMAMUX_SYNC_DMAMUX_CH1
  - LL_DMAMUX_SYNC_DMAMUX_CH2
  - LL_DMAMUX_SYNC_DMAMUX_CH3
  - LL_DMAMUX_SYNC_LPTIM1_OUT

Reference Manual to LL API cross reference:

- CxCR SYNC_ID LL_DMAMUX_GetSyncID

LL_DMAMUX_EnableRequestGen

Function name

__STATIC_INLINE void LL_DMAMUX_EnableRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Enable the Request Generator.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_EnableRequestGen

LL_DMAMUX_DisableRequestGen

Function name

__STATIC_INLINE void LL_DMAMUX_DisableRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Disable the Request Generator.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_DisableRequestGen

LL_DMAMUX_IsEnabledRequestGen

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Check if the Request Generator is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_IsEnabledRequestGen

LL_DMAMUX_SetRequestGenPolarity

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestGenPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t Polarity)

Function description

Set the polarity of the signal on which the DMA request is generated.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
- **Polarity:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_NO_EVENT
  - LL_DMAMUX_REQ_GEN_POL_RISING
  - LL_DMAMUX_REQ_GEN_POL_FALLING
  - LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GPOL LL_DMAMUX_SetRequestGenPolarity

LL_DMAMUX_GetRequestGenPolarity

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestGenPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get the polarity of the signal on which the DMA request is generated.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_GEN_NO_EVENT
  - LL_DMAMUX_REQ_GEN_POL_RISING
  - LL_DMAMUX_REQ_GEN_POL_FALLING
  - LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

Reference Manual to LL API cross reference:

- RGxCR GPOL LL_DMAMUX_GetRequestGenPolarity

LL_DMAMUX_SetGenRequestNb

Function name

__STATIC_INLINE void LL_DMAMUX_SetGenRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestNb)

Function description

Set the number of DMA request that will be autorized after a generation event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
- **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.

Return values

- **None:**

Notes

- This field can only be written when Generator is disabled.

Reference Manual to LL API cross reference:

- RGxCR GNBREQ LL_DMAMUX_SetGenRequestNb

LL_DMAMUX_GetGenRequestNb

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetGenRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get the number of DMA request that will be autorized after a generation event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **Between:** Min_Data = 1 and Max_Data = 32

Reference Manual to LL API cross reference:

- RGxCR GNBREQ LL_DMAMUX_GetGenRequestNb

LL_DMAMUX_SetRequestSignalID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestSignalID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestSignalID)

Function description

Set DMAMUX external Request Signal ID on DMAMUX Request Generation Trigger Event Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
- **RequestSignalID:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_EXTI_LINE0
  - LL_DMAMUX_REQ_GEN_EXTI_LINE1
  - LL_DMAMUX_REQ_GEN_EXTI_LINE2
  - LL_DMAMUX_REQ_GEN_EXTI_LINE3
  - LL_DMAMUX_REQ_GEN_EXTI_LINE4
  - LL_DMAMUX_REQ_GEN_EXTI_LINE5
  - LL_DMAMUX_REQ_GEN_EXTI_LINE6
  - LL_DMAMUX_REQ_GEN_EXTI_LINE7
  - LL_DMAMUX_REQ_GEN_EXTI_LINE8
  - LL_DMAMUX_REQ_GEN_EXTI_LINE9
  - LL_DMAMUX_REQ_GEN_EXTI_LINE10
  - LL_DMAMUX_REQ_GEN_EXTI_LINE11
  - LL_DMAMUX_REQ_GEN_EXTI_LINE12
  - LL_DMAMUX_REQ_GEN_EXTI_LINE13
  - LL_DMAMUX_REQ_GEN_EXTI_LINE14
  - LL_DMAMUX_REQ_GEN_EXTI_LINE15
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH0
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH1
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH2
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH3
  - LL_DMAMUX_REQ_GEN_LPTIM1_OUT

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR SIG_ID LL_DMAMUX_SetRequestSignalID

LL_DMAMUX_GetRequestSignalID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestSignalID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get DMAMUX external Request Signal ID set on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_GEN_EXTI_LINE0
  - LL_DMAMUX_REQ_GEN_EXTI_LINE1
  - LL_DMAMUX_REQ_GEN_EXTI_LINE2
  - LL_DMAMUX_REQ_GEN_EXTI_LINE3
  - LL_DMAMUX_REQ_GEN_EXTI_LINE4
  - LL_DMAMUX_REQ_GEN_EXTI_LINE5
  - LL_DMAMUX_REQ_GEN_EXTI_LINE6
  - LL_DMAMUX_REQ_GEN_EXTI_LINE7
  - LL_DMAMUX_REQ_GEN_EXTI_LINE8
  - LL_DMAMUX_REQ_GEN_EXTI_LINE9
  - LL_DMAMUX_REQ_GEN_EXTI_LINE10
  - LL_DMAMUX_REQ_GEN_EXTI_LINE11
  - LL_DMAMUX_REQ_GEN_EXTI_LINE12
  - LL_DMAMUX_REQ_GEN_EXTI_LINE13
  - LL_DMAMUX_REQ_GEN_EXTI_LINE14
  - LL_DMAMUX_REQ_GEN_EXTI_LINE15
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH0
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH1
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH2
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH3
  - LL_DMAMUX_REQ_GEN_LPTIM1_OUT

Reference Manual to LL API cross reference:

- RGxCR SIG_ID LL_DMAMUX_GetRequestSignalID

LL_DMAMUX_IsActiveFlag_SO0

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 0.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF0 LL_DMAMUX_IsActiveFlag_SO0

LL_DMAMUX_IsActiveFlag_SO1

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 1.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF1 LL_DMAMUX_IsActiveFlag_SO1

LL_DMAMUX_IsActiveFlag_SO2

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 2.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF2 LL_DMAMUX_IsActiveFlag_SO2

LL_DMAMUX_IsActiveFlag_SO3

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 3.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF3 LL_DMAMUX_IsActiveFlag_SO3

LL_DMAMUX_IsActiveFlag_SO4

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO4 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 4.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF4 LL_DMAMUX_IsActiveFlag_SO4

LL_DMAMUX_IsActiveFlag_SO5

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO5 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 5.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF5 LL_DMAMUX_IsActiveFlag_SO5

LL_DMAMUX_IsActiveFlag_SO6

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO6 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 6.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF6 LL_DMAMUX_IsActiveFlag_SO6

LL_DMAMUX_IsActiveFlag_SO7

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO7 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 7.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF7 LL_DMAMUX_IsActiveFlag_SO7

LL_DMAMUX_IsActiveFlag_SO8

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO8 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 8.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF8 LL_DMAMUX_IsActiveFlag_SO8

LL_DMAMUX_IsActiveFlag_SO9

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO9 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 9.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF9 LL_DMAMUX_IsActiveFlag_SO9

LL_DMAMUX_IsActiveFlag_SO10

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO10 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 10.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF10 LL_DMAMUX_IsActiveFlag_SO10

LL_DMAMUX_IsActiveFlag_SO11

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO11 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 11.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF11 LL_DMAMUX_IsActiveFlag_SO11

LL_DMAMUX_IsActiveFlag_SO12

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO12 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 12.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF12 LL_DMAMUX_IsActiveFlag_SO12

LL_DMAMUX_IsActiveFlag_SO13

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO13 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 13.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF13 LL_DMAMUX_IsActiveFlag_SO13

LL_DMAMUX_IsActiveFlag_SO14

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO14 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 14.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF13 LL_DMAMUX_IsActiveFlag_SO14

LL_DMAMUX_IsActiveFlag_SO15

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO15 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 15.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF13 LL_DMAMUX_IsActiveFlag_SO15

LL_DMAMUX_IsActiveFlag_RGO0

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 0 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF0 LL_DMAMUX_IsActiveFlag_RGO0

LL_DMAMUX_IsActiveFlag_RGO1

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 1 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF1 LL_DMAMUX_IsActiveFlag_RGO1

LL_DMAMUX_IsActiveFlag_RGO2

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 2 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF2 LL_DMAMUX_IsActiveFlag_RGO2

LL_DMAMUX_IsActiveFlag_RGO3

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 3 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF3 LL_DMAMUX_IsActiveFlag_RGO3

LL_DMAMUX_ClearFlag_SO0

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 0.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF0 LL_DMAMUX_ClearFlag_SO0

LL_DMAMUX_ClearFlag_SO1

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 1.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF1 LL_DMAMUX_ClearFlag_SO1

LL_DMAMUX_ClearFlag_SO2

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 2.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF2 LL_DMAMUX_ClearFlag_SO2

LL_DMAMUX_ClearFlag_SO3

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 3.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF3 LL_DMAMUX_ClearFlag_SO3

LL_DMAMUX_ClearFlag_SO4

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO4 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 4.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF4 LL_DMAMUX_ClearFlag_SO4

LL_DMAMUX_ClearFlag_SO5

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO5 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 5.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF5 LL_DMAMUX_ClearFlag_SO5

LL_DMAMUX_ClearFlag_SO6

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO6 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 6.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF6 LL_DMAMUX_ClearFlag_SO6

LL_DMAMUX_ClearFlag_SO7

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO7 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 7.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF7 LL_DMAMUX_ClearFlag_SO7

LL_DMAMUX_ClearFlag_SO8

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO8 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 8.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF8 LL_DMAMUX_ClearFlag_SO8

LL_DMAMUX_ClearFlag_SO9

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO9 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 9.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF9 LL_DMAMUX_ClearFlag_SO9

LL_DMAMUX_ClearFlag_SO10

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO10 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 10.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF10 LL_DMAMUX_ClearFlag_SO10

LL_DMAMUX_ClearFlag_SO11

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO11 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 11.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF11 LL_DMAMUX_ClearFlag_SO11

LL_DMAMUX_ClearFlag_SO12

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO12 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 12.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF12 LL_DMAMUX_ClearFlag_SO12

LL_DMAMUX_ClearFlag_SO13

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO13 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 13.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF13 LL_DMAMUX_ClearFlag_SO13

LL_DMAMUX_ClearFlag_SO14

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO14 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 14.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF14 LL_DMAMUX_ClearFlag_SO14

LL_DMAMUX_ClearFlag_SO15

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO15 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 15.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF15 LL_DMAMUX_ClearFlag_SO15

LL_DMAMUX_ClearFlag_RGO0

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 0 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF0 LL_DMAMUX_ClearFlag_RGO0

LL_DMAMUX_ClearFlag_RGO1

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 1 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF1 LL_DMAMUX_ClearFlag_RGO1

LL_DMAMUX_ClearFlag_RGO2

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 2 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF2 LL_DMAMUX_ClearFlag_RGO2

LL_DMAMUX_ClearFlag_RGO3

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 3 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF3 LL_DMAMUX_ClearFlag_RGO3

LL_DMAMUX_EnableIT_SO

Function name

__STATIC_INLINE void LL_DMAMUX_EnableIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the Synchronization Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_EnableIT_SO

LL_DMAMUX_DisableIT_SO

Function name

__STATIC_INLINE void LL_DMAMUX_DisableIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the Synchronization Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_DisableIT_SO

LL_DMAMUX_IsEnabledIT_SO

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the Synchronization Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_IsEnabledIT_SO

LL_DMAMUX_EnableIT_RGO

Function name

__STATIC_INLINE void LL_DMAMUX_EnableIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Enable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_EnableIT_RGO

LL_DMAMUX_DisableIT_RGO

Function name

__STATIC_INLINE void LL_DMAMUX_DisableIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Disable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_DisableIT_RGO

LL_DMAMUX_IsEnabledIT_RGO

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Check if the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_IsEnabledIT_RGO

73.2 DMAMUX Firmware driver defines

The following section lists the various define and macros of the module.

73.2.1 DMAMUX

DMAMUX

DMAMUX Channel

LL_DMAMUX_CHANNEL_0

DMAMUX Channel 0 connected to DMA1 Channel 1

LL_DMAMUX_CHANNEL_1

DMAMUX Channel 1 connected to DMA1 Channel 2

LL_DMAMUX_CHANNEL_2

DMAMUX Channel 2 connected to DMA1 Channel 3

LL_DMAMUX_CHANNEL_3

DMAMUX Channel 3 connected to DMA1 Channel 4

LL_DMAMUX_CHANNEL_4

DMAMUX Channel 4 connected to DMA1 Channel 5

LL_DMAMUX_CHANNEL_5

DMAMUX Channel 5 connected to DMA1 Channel 6

LL_DMAMUX_CHANNEL_6

DMAMUX Channel 6 connected to DMA1 Channel 7

LL_DMAMUX_CHANNEL_7

DMAMUX Channel 7 connected to DMA1 Channel 8

LL_DMAMUX_CHANNEL_8

DMAMUX Channel 8 connected to DMA2 Channel 1

LL_DMAMUX_CHANNEL_9

DMAMUX Channel 9 connected to DMA2 Channel 2

LL_DMAMUX_CHANNEL_10

DMAMUX Channel 10 connected to DMA2 Channel 3

LL_DMAMUX_CHANNEL_11

DMAMUX Channel 11 connected to DMA2 Channel 4

LL_DMAMUX_CHANNEL_12

DMAMUX Channel 12 connected to DMA2 Channel 5

LL_DMAMUX_CHANNEL_13

DMAMUX Channel 13 connected to DMA2 Channel 6

LL_DMAMUX_CHANNEL_14

DMAMUX Channel 14 connected to DMA2 Channel 7

LL_DMAMUX_CHANNEL_15

DMAMUX Channel 15 connected to DMA2 Channel 8

Clear Flags Defines

LL_DMAMUX_CFR_CSOF0

Synchronization Event Overrun Flag Channel 0

LL_DMAMUX_CFR_CSOF1

Synchronization Event Overrun Flag Channel 1

LL_DMAMUX_CFR_CSOF2

Synchronization Event Overrun Flag Channel 2

LL_DMAMUX_CFR_CSOF3

Synchronization Event Overrun Flag Channel 3

LL_DMAMUX_CFR_CSOF4

Synchronization Event Overrun Flag Channel 4

LL_DMAMUX_CFR_CSOF5

Synchronization Event Overrun Flag Channel 5

LL_DMAMUX_CFR_CSOF6

Synchronization Event Overrun Flag Channel 6

LL_DMAMUX_CFR_CSOF7

Synchronization Event Overrun Flag Channel 7

LL_DMAMUX_CFR_CSOF8

Synchronization Event Overrun Flag Channel 8

LL_DMAMUX_CFR_CSOF9

Synchronization Event Overrun Flag Channel 9

LL_DMAMUX_CFR_CSOF10

Synchronization Event Overrun Flag Channel 10

LL_DMAMUX_CFR_CSOF11

Synchronization Event Overrun Flag Channel 11

LL_DMAMUX_CFR_CSOF12

Synchronization Event Overrun Flag Channel 12

LL_DMAMUX_CFR_CSOF13

Synchronization Event Overrun Flag Channel 13

LL_DMAMUX_CFR_CSOF14

Synchronization Event Overrun Flag Channel 14

LL_DMAMUX_CFR_CSOF15

Synchronization Event Overrun Flag Channel 15

LL_DMAMUX_RGCFR_RGCOF0

Request Generator 0 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF1

Request Generator 1 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF2

Request Generator 2 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF3

Request Generator 3 Trigger Event Overrun Flag

Get Flags Defines

LL_DMAMUX_CSR_SOF0

Synchronization Event Overrun Flag Channel 0

LL_DMAMUX_CSR_SOF1

Synchronization Event Overrun Flag Channel 1

LL_DMAMUX_CSR_SOF2

Synchronization Event Overrun Flag Channel 2

LL_DMAMUX_CSR_SOF3

Synchronization Event Overrun Flag Channel 3

LL_DMAMUX_CSR_SOF4

Synchronization Event Overrun Flag Channel 4

LL_DMAMUX_CSR_SOF5

Synchronization Event Overrun Flag Channel 5

LL_DMAMUX_CSR_SOF6

Synchronization Event Overrun Flag Channel 6

LL_DMAMUX_CSR_SOF7

Synchronization Event Overrun Flag Channel 7

LL_DMAMUX_CSR_SOF8

Synchronization Event Overrun Flag Channel 8

LL_DMAMUX_CSR_SOF9

Synchronization Event Overrun Flag Channel 9

LL_DMAMUX_CSR_SOF10

Synchronization Event Overrun Flag Channel 10

LL_DMAMUX_CSR_SOF11

Synchronization Event Overrun Flag Channel 11

LL_DMAMUX_CSR_SOF12

Synchronization Event Overrun Flag Channel 12

LL_DMAMUX_CSR_SOF13

Synchronization Event Overrun Flag Channel 13

LL_DMAMUX_CSR_SOF14

Synchronization Event Overrun Flag Channel 14

LL_DMAMUX_CSR_SOF15

Synchronization Event Overrun Flag Channel 15

LL_DMAMUX_RGSR_RGOF0

Request Generator 0 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF1

Request Generator 1 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF2

Request Generator 2 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF3

Request Generator 3 Trigger Event Overrun Flag

IT Defines

LL_DMAMUX_CCR_SOIE

Synchronization Event Overrun Interrupt

LL_DMAMUX_RGCR_RGOIE

Request Generation Trigger Event Overrun Interrupt

Transfer request

LL_DMAMUX_REQ_MEM2MEM

Memory to memory transfer

LL_DMAMUX_REQ_GENERATOR0

DMAMUX request generator 0

LL_DMAMUX_REQ_GENERATOR1

DMAMUX request generator 1

LL_DMAMUX_REQ_GENERATOR2

DMAMUX request generator 2

LL_DMAMUX_REQ_GENERATOR3

DMAMUX request generator 3

LL_DMAMUX_REQ_ADC1

DMAMUX ADC1 request

LL_DMAMUX_REQ_DAC1_CH1

DMAMUX DAC1 CH1 request

LL_DMAMUX_REQ_DAC1_CH2

DMAMUX DAC1 CH2 request

LL_DMAMUX_REQ_TIM6_UP

DMAMUX TIM6 UP request

LL_DMAMUX_REQ_TIM7_UP

DMAMUX TIM7 UP request

LL_DMAMUX_REQ_SPI1_RX

DMAMUX SPI1 RX request

LL_DMAMUX_REQ_SPI1_TX

DMAMUX SPI1 TX request

LL_DMAMUX_REQ_SPI2_RX

DMAMUX SPI2 RX request

LL_DMAMUX_REQ_SPI2_TX

DMAMUX SPI2 TX request

LL_DMAMUX_REQ_SPI3_RX

DMAMUX SPI3 RX request

LL_DMAMUX_REQ_SPI3_TX

DMAMUX SPI3 TX request

LL_DMAMUX_REQ_I2C1_RX

DMAMUX I2C1 RX request

LL_DMAMUX_REQ_I2C1_TX

DMAMUX I2C1 TX request

LL_DMAMUX_REQ_I2C2_RX

DMAMUX I2C2 RX request

LL_DMAMUX_REQ_I2C2_TX

DMAMUX I2C2 TX request

LL_DMAMUX_REQ_I2C3_RX

DMAMUX I2C3 RX request

LL_DMAMUX_REQ_I2C3_TX

DMAMUX I2C3 TX request

LL_DMAMUX_REQ_I2C4_RX

DMAMUX I2C4 RX request

LL_DMAMUX_REQ_I2C4_TX

DMAMUX I2C4 TX request

LL_DMAMUX_REQ_USART1_RX

DMAMUX USART1 RX request

LL_DMAMUX_REQ_USART1_TX

DMAMUX USART1 TX request

LL_DMAMUX_REQ_USART2_RX

DMAMUX USART2 RX request

LL_DMAMUX_REQ_USART2_TX

DMAMUX USART2 TX request

LL_DMAMUX_REQ_USART3_RX

DMAMUX USART3 RX request

LL_DMAMUX_REQ_USART3_TX

DMAMUX USART3 TX request

LL_DMAMUX_REQ_UART4_RX

DMAMUX UART4 RX request

LL_DMAMUX_REQ_UART4_TX

DMAMUX UART4 TX request

LL_DMAMUX_REQ_UART5_RX

DMAMUX UART5 RX request

LL_DMAMUX_REQ_UART5_TX

DMAMUX UART5 TX request

LL_DMAMUX_REQ_LPUART1_RX

DMAMUX LPUART1 RX request

LL_DMAMUX_REQ_LPUART1_TX

DMAMUX LPUART1 TX request

LL_DMAMUX_REQ_ADC2

DMAMUX ADC2 request

LL_DMAMUX_REQ_ADC3

DMAMUX ADC3 request

LL_DMAMUX_REQ_ADC4

DMAMUX ADC4 request

LL_DMAMUX_REQ_ADC5

DMAMUX ADC5 request

LL_DMAMUX_REQ_QSPI

DMAMUX QSPI request

LL_DMAMUX_REQ_DAC2_CH1

DMAMUX DAC2 CH1 request

LL_DMAMUX_REQ_TIM1_CH1

DMAMUX TIM1 CH1 request

LL_DMAMUX_REQ_TIM1_CH2

DMAMUX TIM1 CH2 request

LL_DMAMUX_REQ_TIM1_CH3

DMAMUX TIM1 CH3 request

LL_DMAMUX_REQ_TIM1_CH4

DMAMUX TIM1 CH4 request

LL_DMAMUX_REQ_TIM1_UP

DMAMUX TIM1 UP request

LL_DMAMUX_REQ_TIM1_TRIG

DMAMUX TIM1 TRIG request

LL_DMAMUX_REQ_TIM1_COM

DMAMUX TIM1 COM request

LL_DMAMUX_REQ_TIM8_CH1

DMAMUX TIM8 CH1 request

LL_DMAMUX_REQ_TIM8_CH2

DMAMUX TIM8 CH2 request

LL_DMAMUX_REQ_TIM8_CH3

DMAMUX TIM8 CH3 request

LL_DMAMUX_REQ_TIM8_CH4

DMAMUX TIM8 CH4 request

LL_DMAMUX_REQ_TIM8_UP

DMAMUX TIM8 UP request

LL_DMAMUX_REQ_TIM8_TRIG

DMAMUX TIM8 TRIG request

LL_DMAMUX_REQ_TIM8_COM

DMAMUX TIM8 COM request

LL_DMAMUX_REQ_TIM2_CH1

DMAMUX TIM2 CH1 request

LL_DMAMUX_REQ_TIM2_CH2

DMAMUX TIM2 CH2 request

LL_DMAMUX_REQ_TIM2_CH3

DMAMUX TIM2 CH3 request

LL_DMAMUX_REQ_TIM2_CH4

DMAMUX TIM2 CH4 request

LL_DMAMUX_REQ_TIM2_UP

DMAMUX TIM2 UP request

LL_DMAMUX_REQ_TIM3_CH1

DMAMUX TIM3 CH1 request

LL_DMAMUX_REQ_TIM3_CH2

DMAMUX TIM3 CH2 request

LL_DMAMUX_REQ_TIM3_CH3

DMAMUX TIM3 CH3 request

LL_DMAMUX_REQ_TIM3_CH4

DMAMUX TIM3 CH4 request

LL_DMAMUX_REQ_TIM3_UP

DMAMUX TIM3 UP request

LL_DMAMUX_REQ_TIM3_TRIG

DMAMUX TIM3 TRIG request

LL_DMAMUX_REQ_TIM4_CH1

DMAMUX TIM4 CH1 request

LL_DMAMUX_REQ_TIM4_CH2

DMAMUX TIM4 CH2 request

LL_DMAMUX_REQ_TIM4_CH3

DMAMUX TIM4 CH3 request

LL_DMAMUX_REQ_TIM4_CH4

DMAMUX TIM4 CH4 request

LL_DMAMUX_REQ_TIM4_UP

DMAMUX TIM4 UP request

LL_DMAMUX_REQ_TIM5_CH1

DMAMUX TIM5 CH1 request

LL_DMAMUX_REQ_TIM5_CH2

DMAMUX TIM5 CH2 request

LL_DMAMUX_REQ_TIM5_CH3

DMAMUX TIM5 CH3 request

LL_DMAMUX_REQ_TIM5_CH4

DMAMUX TIM5 CH4 request

LL_DMAMUX_REQ_TIM5_UP

DMAMUX TIM5 UP request

LL_DMAMUX_REQ_TIM5_TRIG

DMAMUX TIM5 TRIG request

LL_DMAMUX_REQ_TIM15_CH1

DMAMUX TIM15 CH1 request

LL_DMAMUX_REQ_TIM15_UP

DMAMUX TIM15 UP request

LL_DMAMUX_REQ_TIM15_TRIG

DMAMUX TIM15 TRIG request

LL_DMAMUX_REQ_TIM15_COM

DMAMUX TIM15 COM request

LL_DMAMUX_REQ_TIM16_CH1

DMAMUX TIM16 CH1 request

LL_DMAMUX_REQ_TIM16_UP

DMAMUX TIM16 UP request

LL_DMAMUX_REQ_TIM17_CH1

DMAMUX TIM17 CH1 request

LL_DMAMUX_REQ_TIM17_UP

DMAMUX TIM17 UP request

LL_DMAMUX_REQ_TIM20_CH1

DMAMUX TIM20 CH1 request

LL_DMAMUX_REQ_TIM20_CH2

DMAMUX TIM20 CH2 request

LL_DMAMUX_REQ_TIM20_CH3

DMAMUX TIM20 CH3 request

LL_DMAMUX_REQ_TIM20_CH4

DMAMUX TIM20 CH4 request

LL_DMAMUX_REQ_TIM20_UP

DMAMUX TIM20 UP request

LL_DMAMUX_REQ_AES_IN

DMAMUX AES_IN request

LL_DMAMUX_REQ_AES_OUT

DMAMUX AES_OUT request

LL_DMAMUX_REQ_TIM20_TRIG

DMAMUX TIM20 TRIG request

LL_DMAMUX_REQ_TIM20_COM

DMAMUX TIM20 COM request

LL_DMAMUX_REQ_HRTIM1_M

DMAMUX HRTIM M request

LL_DMAMUX_REQ_HRTIM1_A

DMAMUX HRTIM A request

LL_DMAMUX_REQ_HRTIM1_B

DMAMUX HRTIM B request

LL_DMAMUX_REQ_HRTIM1_C

DMAMUX HRTIM C request

LL_DMAMUX_REQ_HRTIM1_D

DMAMUX HRTIM D request

LL_DMAMUX_REQ_HRTIM1_E

DMAMUX HRTIM E request

LL_DMAMUX_REQ_HRTIM1_F

DMAMUX HRTIM F request

LL_DMAMUX_REQ_DAC3_CH1

DMAMUX DAC3 CH1 request

LL_DMAMUX_REQ_DAC3_CH2

DMAMUX DAC3 CH2 request

LL_DMAMUX_REQ_DAC4_CH1

DMAMUX DAC4 CH1 request

LL_DMAMUX_REQ_DAC4_CH2

DMAMUX DAC4 CH2 request

LL_DMAMUX_REQ_SPI4_RX

DMAMUX SPI4 RX request

LL_DMAMUX_REQ_SPI4_TX

DMAMUX SPI4 TX request

LL_DMAMUX_REQ_SAI1_A

DMAMUX SAI1 A request

LL_DMAMUX_REQ_SAI1_B

DMAMUX SAI1 B request

LL_DMAMUX_REQ_FMAC_WRITE

DMAMUX FMAC WRITE request

LL_DMAMUX_REQ_FMAC_READ

DMAMUX FMAC READ request

LL_DMAMUX_REQ_CORDIC_WRITE

DMAMUX CORDIC WRITE request

LL_DMAMUX_REQ_CORDIC_READ

DMAMUX CORDIC READ request

LL_DMAMUX_REQ_UCPD1_RX

DMAMUX USBPD1_RX request

LL_DMAMUX_REQ_UCPD1_TX

DMAMUX USBPD1_TX request

External Request Signal Generation

LL_DMAMUX_REQ_GEN_EXTI_LINE0

Request signal generation from EXTI Line0

LL_DMAMUX_REQ_GEN_EXTI_LINE1

Request signal generation from EXTI Line1

LL_DMAMUX_REQ_GEN_EXTI_LINE2

Request signal generation from EXTI Line2

LL_DMAMUX_REQ_GEN_EXTI_LINE3

Request signal generation from EXTI Line3

LL_DMAMUX_REQ_GEN_EXTI_LINE4

Request signal generation from EXTI Line4

LL_DMAMUX_REQ_GEN_EXTI_LINE5

Request signal generation from EXTI Line5

LL_DMAMUX_REQ_GEN_EXTI_LINE6

Request signal generation from EXTI Line6

LL_DMAMUX_REQ_GEN_EXTI_LINE7

Request signal generation from EXTI Line7

LL_DMAMUX_REQ_GEN_EXTI_LINE8

Request signal generation from EXTI Line8

LL_DMAMUX_REQ_GEN_EXTI_LINE9

Request signal generation from EXTI Line9

LL_DMAMUX_REQ_GEN_EXTI_LINE10

Request signal generation from EXTI Line10

LL_DMAMUX_REQ_GEN_EXTI_LINE11

Request signal generation from EXTI Line11

LL_DMAMUX_REQ_GEN_EXTI_LINE12

Request signal generation from EXTI Line12

LL_DMAMUX_REQ_GEN_EXTI_LINE13

Request signal generation from EXTI Line13

LL_DMAMUX_REQ_GEN_EXTI_LINE14

Request signal generation from EXTI Line14

LL_DMAMUX_REQ_GEN_EXTI_LINE15

Request signal generation from EXTI Line15

LL_DMAMUX_REQ_GEN_DMAMUX_CH0

Request signal generation from DMAMUX channel0 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH1

Request signal generation from DMAMUX channel1 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH2

Request signal generation from DMAMUX channel2 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH3

Request signal generation from DMAMUX channel3 Event

LL_DMAMUX_REQ_GEN_LPTIM1_OUT

Request signal generation from LPTIM1 Ouput

Request Generator Channel

LL_DMAMUX_REQ_GEN_0

LL_DMAMUX_REQ_GEN_1

LL_DMAMUX_REQ_GEN_2

LL_DMAMUX_REQ_GEN_3

External Request Signal Generation Polarity

LL_DMAMUX_REQ_GEN_NO_EVENT

No external DMA request generation

LL_DMAMUX_REQ_GEN_POL_RISING

External DMA request generation on event on rising edge

LL_DMAMUX_REQ_GEN_POL_FALLING

External DMA request generation on event on falling edge

LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

External DMA request generation on rising and falling edge

Synchronization Signal Event

LL_DMAMUX_SYNC_EXTI_LINE0

Synchronization signal from EXTI Line0

LL_DMAMUX_SYNC_EXTI_LINE1

Synchronization signal from EXTI Line1

LL_DMAMUX_SYNC_EXTI_LINE2

Synchronization signal from EXTI Line2

LL_DMAMUX_SYNC_EXTI_LINE3

Synchronization signal from EXTI Line3

LL_DMAMUX_SYNC_EXTI_LINE4

Synchronization signal from EXTI Line4

LL_DMAMUX_SYNC_EXTI_LINE5

Synchronization signal from EXTI Line5

LL_DMAMUX_SYNC_EXTI_LINE6

Synchronization signal from EXTI Line6

LL_DMAMUX_SYNC_EXTI_LINE7

Synchronization signal from EXTI Line7

LL_DMAMUX_SYNC_EXTI_LINE8

Synchronization signal from EXTI Line8

LL_DMAMUX_SYNC_EXTI_LINE9

Synchronization signal from EXTI Line9

LL_DMAMUX_SYNC_EXTI_LINE10

Synchronization signal from EXTI Line10

LL_DMAMUX_SYNC_EXTI_LINE11

Synchronization signal from EXTI Line11

LL_DMAMUX_SYNC_EXTI_LINE12

Synchronization signal from EXTI Line12

LL_DMAMUX_SYNC_EXTI_LINE13

Synchronization signal from EXTI Line13

LL_DMAMUX_SYNC_EXTI_LINE14

Synchronization signal from EXTI Line14

LL_DMAMUX_SYNC_EXTI_LINE15

Synchronization signal from EXTI Line15

LL_DMAMUX_SYNC_DMAMUX_CH0

Synchronization signal from DMAMUX channel0 Event

LL_DMAMUX_SYNC_DMAMUX_CH1

Synchronization signal from DMAMUX channel1 Event

LL_DMAMUX_SYNC_DMAMUX_CH2

Synchronization signal from DMAMUX channel2 Event

LL_DMAMUX_SYNC_DMAMUX_CH3

Synchronization signal from DMAMUX channel3 Event

LL_DMAMUX_SYNC_LPTIM1_OUT

Synchronization signal from LPTIM1 Ouput

Synchronization Signal Polarity

LL_DMAMUX_SYNC_NO_EVENT

All requests are blocked

LL_DMAMUX_SYNC_POL_RISING

Synchronization on event on rising edge

LL_DMAMUX_SYNC_POL_FALLING

Synchronization on event on falling edge

LL_DMAMUX_SYNC_POL_RISING_FALLING

Synchronization on event on rising and falling edge

Common Write and read registers macros

LL_DMAMUX_WriteReg

Description:

- Write a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMAMUX_ReadReg

Description:

- Read a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be read

Return value:

- Register: value

74 LL DMA Generic Driver

74.1 DMA Firmware driver registers structures

74.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32g4xx_ll_dma.h

Data Fields

- ***uint32_t PeriphOrM2MSrcAddress***
- ***uint32_t MemoryOrM2MDstAddress***
- ***uint32_t Direction***
- ***uint32_t Mode***
- ***uint32_t PeriphOrM2MSrcIncMode***
- ***uint32_t MemoryOrM2MDstIncMode***
- ***uint32_t PeriphOrM2MSrcDataSize***
- ***uint32_t MemoryOrM2MDstDataSize***
- ***uint32_t NbData***
- ***uint32_t PeriphRequest***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcAddress***

  Specifies the peripheral base address for DMA transfer or as Source base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstAddress***

  Specifies the memory base address for DMA transfer or as Destination base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitTypeDef::Direction***

  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_LL_EC_DIRECTION***This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::Mode***

  Specifies the normal or circular operation mode. This parameter can be a value of ***DMA_LL_EC_MODE***

  Note:

  – : The circular buffer mode cannot be used if the memory to memory data transfer direction is configured on the selected Channel

  This feature can be modified afterwards using unitary function **LL_DMA_SetMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcIncMode***

  Specifies whether the Peripheral address or Source address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_PERIPH***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstIncMode***

  Specifies whether the Memory address or Destination address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_MEMORY***This feature can be modified afterwards using unitary function **LL_DMA_SetMemoryIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcDataSize***

  Specifies the Peripheral data size alignment or Source data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_PDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphSize()**.

- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstDataSize***

  Specifies the Memory data size alignment or Destination data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_MDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetMemorySize()**.
- ***uint32_t LL_DMA_InitTypeDef::NbData***

  Specifies the number of data to transfer, in data unit. The data unit is equal to the source buffer configuration set in PeripheralSize or MemorySize parameters depending in the transfer direction. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFFThis feature can be modified afterwards using unitary function **LL_DMA_SetDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphRequest***

  Specifies the peripheral request. This parameter can be a value of ***DMAMUX_LL_EC_REQUEST***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***

  Specifies the channel priority level. This parameter can be a value of ***DMA_LL_EC_PRIORITY***This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.

74.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

74.2.1 Detailed description of functions

LL_DMA_EnableChannel

Function name

__STATIC_INLINE void LL_DMA_EnableChannel (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name

__STATIC_INLINE void LL_DMA_DisableChannel (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel is enabled or disabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all parameters link to DMA transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY
  - LL_DMA_MODE_NORMAL or LL_DMA_MODE_CIRCULAR
  - LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT
  - LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT
  - LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or LL_DMA_PDATAALIGN_WORD
  - LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or LL_DMA_MDATAALIGN_WORD
  - LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR DIR LL_DMA_ConfigTransfer
- CCR MEM2MEM LL_DMA_ConfigTransfer
- CCR CIRC LL_DMA_ConfigTransfer
- CCR PINC LL_DMA_ConfigTransfer
- CCR MINC LL_DMA_ConfigTransfer
- CCR PSIZE LL_DMA_ConfigTransfer
- CCR MSIZE LL_DMA_ConfigTransfer
- CCR PL LL_DMA_ConfigTransfer

LL_DMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_DMA_SetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description

Set Data transfer direction (read from peripheral or from memory).

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR DIR LL_DMA_SetDataTransferDirection
- CCR MEM2MEM LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Data transfer direction (read from peripheral or from memory).

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Reference Manual to LL API cross reference:

- CCR DIR LL_DMA_GetDataTransferDirection
- CCR MEM2MEM LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name

__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)

Function description

Set DMA mode circular or normal.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **Mode:** This parameter can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR

Return values

- **None:**

Notes

- The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.

Reference Manual to LL API cross reference:

- CCR CIRC LL_DMA_SetMode

LL_DMA_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get DMA mode circular or normal.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR

Reference Manual to LL API cross reference:

- CCR CIRC LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name

__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)

Function description

Set Peripheral increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values:
  - LL_DMA_PERIPH_INCREMENT
  - LL_DMA_PERIPH_NOINCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PINC LL_DMA_SetPeriphIncMode

LL_DMA_GetPeriphIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Peripheral increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PERIPH_INCREMENT
  - LL_DMA_PERIPH_NOINCREMENT

Reference Manual to LL API cross reference:

- CCR PINC LL_DMA_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name

__STATIC_INLINE void LL_DMA_SetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)

Function description

Set Memory increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **MemoryOrM2MDstIncMode:** This parameter can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR MINC LL_DMA_SetMemoryIncMode

LL_DMA_GetMemoryIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Memory increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Reference Manual to LL API cross reference:

- CCR MINC LL_DMA_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name

__STATIC_INLINE void LL_DMA_SetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)

Function description

Set Peripheral size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **PeriphOrM2MSrcDataSize:** This parameter can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Peripheral size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name

__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)

Function description

Set Memory size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **MemoryOrM2MDstDataSize:** This parameter can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Memory size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CCR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)

Function description

Set Channel priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **Priority:** This parameter can be one of the following values:
  - LL_DMA_PRIORITY_LOW
  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PL LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Channel priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PRIORITY_LOW
  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Reference Manual to LL API cross reference:

- CCR PL LL_DMA_GetChannelPriorityLevel

LL_DMA_SetDataLength

Function name

__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t NbData)

Function description

Set Number of data to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF

Return values

- **None:**

Notes

- This action has no effect if channel is enabled.

Reference Manual to LL API cross reference:

- CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Number of data to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.

Reference Manual to LL API cross reference:

- CNDTR NDT LL_DMA_GetDataLength

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)

Function description

Configure the Source and Destination addresses.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Notes

- This API must not be called when the DMA channel is enabled.
- Each IP using DMA provides an API to get directly the register adress (LL_PPP_DMA_GetRegAddr).

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_ConfigAddresses
- CMAR MA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name

__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name

__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)

Function description

Set the Peripheral address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **PeriphAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Memory address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Peripheral address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA channel is enabled.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get the Memory to Memory Source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get the Memory to Memory Destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_GetM2MDstAddress

LL_DMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_DMA_SetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphRequest)

Function description

Set DMA request for DMA instance on Channel x.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

- **PeriphRequest:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_DAC1_CH1
  - LL_DMAMUX_REQ_DAC1_CH2
  - LL_DMAMUX_REQ_TIM6_UP
  - LL_DMAMUX_REQ_TIM7_UP
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX
  - LL_DMAMUX_REQ_SPI2_TX
  - LL_DMAMUX_REQ_SPI3_RX
  - LL_DMAMUX_REQ_SPI3_TX
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX
  - LL_DMAMUX_REQ_I2C2_TX
  - LL_DMAMUX_REQ_I2C3_RX
  - LL_DMAMUX_REQ_I2C3_TX (*)
  - LL_DMAMUX_REQ_I2C4_RX (*)
  - LL_DMAMUX_REQ_I2C4_TX
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX
  - LL_DMAMUX_REQ_USART3_TX
  - LL_DMAMUX_REQ_UART4_RX
  - LL_DMAMUX_REQ_UART4_TX
  - LL_DMAMUX_REQ_UART5_RX (*)
  - LL_DMAMUX_REQ_UART5_TX (*)
  - LL_DMAMUX_REQ_LPUART1_RX
  - LL_DMAMUX_REQ_LPUART1_TX
  - LL_DMAMUX_REQ_ADC2
  - LL_DMAMUX_REQ_ADC3 (*)
  - LL_DMAMUX_REQ_ADC4 (*)
  - LL_DMAMUX_REQ_ADC5 (*)
  - LL_DMAMUX_REQ_QSPI (*)
  - LL_DMAMUX_REQ_DAC2_CH1 (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM1_TRIG
  - LL_DMAMUX_REQ_TIM1_COM
  - LL_DMAMUX_REQ_TIM8_CH1
  - LL_DMAMUX_REQ_TIM8_CH2

Return values

- **None:**

Notes

- Please refer to Reference Manual to get the available mapping of Request value link to Channel Selection.

Reference Manual to LL API cross reference:

- CSELR C1S LL_DMA_SetPeriphRequest
- CSELR C2S LL_DMA_SetPeriphRequest
- CSELR C3S LL_DMA_SetPeriphRequest
- CSELR C4S LL_DMA_SetPeriphRequest
- CSELR C5S LL_DMA_SetPeriphRequest
- CSELR C6S LL_DMA_SetPeriphRequest
- CSELR C7S LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get DMA request for DMA instance on Channel x.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_DAC1_CH1
  - LL_DMAMUX_REQ_DAC1_CH2
  - LL_DMAMUX_REQ_TIM6_UP
  - LL_DMAMUX_REQ_TIM7_UP
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX
  - LL_DMAMUX_REQ_SPI2_TX
  - LL_DMAMUX_REQ_SPI3_RX
  - LL_DMAMUX_REQ_SPI3_TX
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX
  - LL_DMAMUX_REQ_I2C2_TX
  - LL_DMAMUX_REQ_I2C3_RX
  - LL_DMAMUX_REQ_I2C3_TX (*)
  - LL_DMAMUX_REQ_I2C4_RX (*)
  - LL_DMAMUX_REQ_I2C4_TX
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX
  - LL_DMAMUX_REQ_USART3_TX
  - LL_DMAMUX_REQ_UART4_RX
  - LL_DMAMUX_REQ_UART4_TX
  - LL_DMAMUX_REQ_UART5_RX (*)
  - LL_DMAMUX_REQ_UART5_TX (*)
  - LL_DMAMUX_REQ_LPUART1_RX
  - LL_DMAMUX_REQ_LPUART1_TX
  - LL_DMAMUX_REQ_ADC2
  - LL_DMAMUX_REQ_ADC3 (*)
  - LL_DMAMUX_REQ_ADC4 (*)
  - LL_DMAMUX_REQ_ADC5 (*)
  - LL_DMAMUX_REQ_QSPI (*)
  - LL_DMAMUX_REQ_DAC2_CH1 (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM1_TRIG
  - LL_DMAMUX_REQ_TIM1_COM
  - LL_DMAMUX_REQ_TIM8_CH1

Reference Manual to LL API cross reference:

- CSELR C1S LL_DMA_GetPeriphRequest
- CSELR C2S LL_DMA_GetPeriphRequest
- CSELR C3S LL_DMA_GetPeriphRequest
- CSELR C4S LL_DMA_GetPeriphRequest
- CSELR C5S LL_DMA_GetPeriphRequest
- CSELR C6S LL_DMA_GetPeriphRequest
- CSELR C7S LL_DMA_GetPeriphRequest

LL_DMA_IsActiveFlag_GI1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI1 (DMA_TypeDef * DMAx)

Function description

Get Channel 1 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF1 LL_DMA_IsActiveFlag_GI1

LL_DMA_IsActiveFlag_GI2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI2 (DMA_TypeDef * DMAx)

Function description

Get Channel 2 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF2 LL_DMA_IsActiveFlag_GI2

LL_DMA_IsActiveFlag_GI3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI3 (DMA_TypeDef * DMAx)

Function description

Get Channel 3 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF3 LL_DMA_IsActiveFlag_GI3

LL_DMA_IsActiveFlag_GI4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4 (DMA_TypeDef * DMAx)

Function description

Get Channel 4 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF4 LL_DMA_IsActiveFlag_GI4

LL_DMA_IsActiveFlag_GI5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5 (DMA_TypeDef * DMAx)

Function description

Get Channel 5 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF5 LL_DMA_IsActiveFlag_GI5

LL_DMA_IsActiveFlag_GI6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6 (DMA_TypeDef * DMAx)

Function description

Get Channel 6 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF6 LL_DMA_IsActiveFlag_GI6

LL_DMA_IsActiveFlag_GI7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7 (DMA_TypeDef * DMAx)

Function description

Get Channel 7 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF7 LL_DMA_IsActiveFlag_GI7

LL_DMA_IsActiveFlag_GI8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF8 LL_DMA_IsActiveFlag_GI8

LL_DMA_IsActiveFlag_TC1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)

Function description

Get Channel 1 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (DMA_TypeDef * DMAx)

Function description

Get Channel 2 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (DMA_TypeDef * DMAx)

Function description

Get Channel 3 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (DMA_TypeDef * DMAx)

Function description

Get Channel 4 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (DMA_TypeDef * DMAx)

Function description

Get Channel 5 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF5 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (DMA_TypeDef * DMAx)

Function description

Get Channel 6 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (DMA_TypeDef * DMAx)

Function description

Get Channel 7 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_TC8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF8 LL_DMA_IsActiveFlag_TC8

LL_DMA_IsActiveFlag_HT1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)

Function description

Get Channel 1 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (DMA_TypeDef * DMAx)

Function description

Get Channel 2 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (DMA_TypeDef * DMAx)

Function description

Get Channel 3 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (DMA_TypeDef * DMAx)

Function description

Get Channel 4 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF4 LL_DMA_IsActiveFlag_HT4

LL_DMA_IsActiveFlag_HT5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (DMA_TypeDef * DMAx)

Function description

Get Channel 5 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF5 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)

Function description

Get Channel 6 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (DMA_TypeDef * DMAx)

Function description

Get Channel 7 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_HT8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF8 LL_DMA_IsActiveFlag_HT8

LL_DMA_IsActiveFlag_TE1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)

Function description

Get Channel 1 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF1 LL_DMA_IsActiveFlag_TE1

LL_DMA_IsActiveFlag_TE2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (DMA_TypeDef * DMAx)

Function description

Get Channel 2 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF2 LL_DMA_IsActiveFlag_TE2

LL_DMA_IsActiveFlag_TE3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (DMA_TypeDef * DMAx)

Function description

Get Channel 3 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF3 LL_DMA_IsActiveFlag_TE3

LL_DMA_IsActiveFlag_TE4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (DMA_TypeDef * DMAx)

Function description

Get Channel 4 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF4 LL_DMA_IsActiveFlag_TE4

LL_DMA_IsActiveFlag_TE5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (DMA_TypeDef * DMAx)

Function description

Get Channel 5 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF5 LL_DMA_IsActiveFlag_TE5

LL_DMA_IsActiveFlag_TE6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)

Function description

Get Channel 6 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF6 LL_DMA_IsActiveFlag_TE6

LL_DMA_IsActiveFlag_TE7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (DMA_TypeDef * DMAx)

Function description

Get Channel 7 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_IsActiveFlag_TE8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF8 LL_DMA_IsActiveFlag_TE8

LL_DMA_ClearFlag_GI1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF1 LL_DMA_ClearFlag_GI1

LL_DMA_ClearFlag_GI2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF2 LL_DMA_ClearFlag_GI2

LL_DMA_ClearFlag_GI3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF3 LL_DMA_ClearFlag_GI3

LL_DMA_ClearFlag_GI4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF4 LL_DMA_ClearFlag_GI4

LL_DMA_ClearFlag_GI5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF5 LL_DMA_ClearFlag_GI5

LL_DMA_ClearFlag_GI6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF6 LL_DMA_ClearFlag_GI6

LL_DMA_ClearFlag_GI7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_GI8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF8 LL_DMA_ClearFlag_GI8

LL_DMA_ClearFlag_TC1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_TC8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF8 LL_DMA_ClearFlag_TC8

LL_DMA_ClearFlag_HT1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_HT8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF8 LL_DMA_ClearFlag_HT8

LL_DMA_ClearFlag_TE1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_ClearFlag_TE8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF8 LL_DMA_ClearFlag_TE8

LL_DMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable Transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_EnableIT_HT

Function name

__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable Half transfer interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable Transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TEIE LL_DMA_EnableIT_TE

LL_DMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable Transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_DisableIT_TC

LL_DMA_DisableIT_HT

Function name

__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable Half transfer interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable Transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TEIE LL_DMA_DisableIT_TE

LL_DMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if Transfer complete Interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_IsEnabledIT_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if Half transfer Interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if Transfer error Interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR TEIE LL_DMA_IsEnabledIT_TE

LL_DMA_Init

Function name

uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Initialize the DMA registers according to the specified parameters in DMA_InitStruct.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*) (*) Not on all G4 devices
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are initialized
  - ERROR: Not applicable

Notes

- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name

uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA registers to their default reset values.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7 (*)
  - LL_DMA_CHANNEL_8 (*)
  - LL_DMA_CHANNEL_ALL (*) Not on all G4 devices

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are de-initialized
  - ERROR: DMA registers are not de-initialized

LL_DMA_StructInit

Function name

void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Set each LL_DMA_InitTypeDef field to default value.

Parameters

- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values

- **None:**

74.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

74.3.1 DMA

DMA

CHANNEL

LL_DMA_CHANNEL_1

DMA Channel 1

LL_DMA_CHANNEL_2

DMA Channel 2

LL_DMA_CHANNEL_3

DMA Channel 3

LL_DMA_CHANNEL_4

DMA Channel 4

LL_DMA_CHANNEL_5

DMA Channel 5

LL_DMA_CHANNEL_6

DMA Channel 6

LL_DMA_CHANNEL_7

DMA Channel 7

LL_DMA_CHANNEL_8

DMA Channel 8

LL_DMA_CHANNEL_ALL

DMA Channel all (used only for function

Clear Flags Defines

LL_DMA_IFCR_CGIF1

Channel 1 global flag

LL_DMA_IFCR_CTCIF1

Channel 1 transfer complete flag

LL_DMA_IFCR_CHTIF1

Channel 1 half transfer flag

LL_DMA_IFCR_CTEIF1

Channel 1 transfer error flag

LL_DMA_IFCR_CGIF2

Channel 2 global flag

LL_DMA_IFCR_CTCIF2

Channel 2 transfer complete flag

LL_DMA_IFCR_CHTIF2

Channel 2 half transfer flag

LL_DMA_IFCR_CTEIF2

Channel 2 transfer error flag

LL_DMA_IFCR_CGIF3

Channel 3 global flag

LL_DMA_IFCR_CTCIF3

Channel 3 transfer complete flag

LL_DMA_IFCR_CHTIF3

Channel 3 half transfer flag

LL_DMA_IFCR_CTEIF3

Channel 3 transfer error flag

LL_DMA_IFCR_CGIF4

Channel 4 global flag

LL_DMA_IFCR_CTCIF4

Channel 4 transfer complete flag

LL_DMA_IFCR_CHTIF4

Channel 4 half transfer flag

LL_DMA_IFCR_CTEIF4

Channel 4 transfer error flag

LL_DMA_IFCR_CGIF5

Channel 5 global flag

LL_DMA_IFCR_CTCIF5

Channel 5 transfer complete flag

LL_DMA_IFCR_CHTIF5

Channel 5 half transfer flag

LL_DMA_IFCR_CTEIF5

Channel 5 transfer error flag

LL_DMA_IFCR_CGIF6

Channel 6 global flag

LL_DMA_IFCR_CTCIF6

Channel 6 transfer complete flag

LL_DMA_IFCR_CHTIF6

Channel 6 half transfer flag

LL_DMA_IFCR_CTEIF6

Channel 6 transfer error flag

LL_DMA_IFCR_CGIF7

Channel 7 global flag

LL_DMA_IFCR_CTCIF7

Channel 7 transfer complete flag

LL_DMA_IFCR_CHTIF7

Channel 7 half transfer flag

LL_DMA_IFCR_CTEIF7

Channel 7 transfer error flag

LL_DMA_IFCR_CGIF8

Channel 8 global flag

LL_DMA_IFCR_CTCIF8

Channel 8 transfer complete flag

LL_DMA_IFCR_CHTIF8

Channel 8 half transfer flag

LL_DMA_IFCR_CTEIF8

Channel 8 transfer error flag

Transfer Direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_DMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

Get Flags Defines

LL_DMA_ISR_GIF1

Channel 1 global flag

LL_DMA_ISR_TCIF1

Channel 1 transfer complete flag

LL_DMA_ISR_HTIF1

Channel 1 half transfer flag

LL_DMA_ISR_TEIF1

Channel 1 transfer error flag

LL_DMA_ISR_GIF2

Channel 2 global flag

LL_DMA_ISR_TCIF2

Channel 2 transfer complete flag

LL_DMA_ISR_HTIF2

Channel 2 half transfer flag

LL_DMA_ISR_TEIF2

Channel 2 transfer error flag

LL_DMA_ISR_GIF3

Channel 3 global flag

LL_DMA_ISR_TCIF3

Channel 3 transfer complete flag

LL_DMA_ISR_HTIF3

Channel 3 half transfer flag

LL_DMA_ISR_TEIF3

Channel 3 transfer error flag

LL_DMA_ISR_GIF4

Channel 4 global flag

LL_DMA_ISR_TCIF4

Channel 4 transfer complete flag

LL_DMA_ISR_HTIF4

Channel 4 half transfer flag

LL_DMA_ISR_TEIF4

Channel 4 transfer error flag

LL_DMA_ISR_GIF5

Channel 5 global flag

LL_DMA_ISR_TCIF5

Channel 5 transfer complete flag

LL_DMA_ISR_HTIF5

Channel 5 half transfer flag

LL_DMA_ISR_TEIF5

Channel 5 transfer error flag

LL_DMA_ISR_GIF6

Channel 6 global flag

LL_DMA_ISR_TCIF6

Channel 6 transfer complete flag

LL_DMA_ISR_HTIF6

Channel 6 half transfer flag

LL_DMA_ISR_TEIF6

Channel 6 transfer error flag

LL_DMA_ISR_GIF7

Channel 7 global flag

LL_DMA_ISR_TCIF7

Channel 7 transfer complete flag

LL_DMA_ISR_HTIF7

Channel 7 half transfer flag

LL_DMA_ISR_TEIF7

Channel 7 transfer error flag

LL_DMA_ISR_GIF8

Channel 8 global flag

LL_DMA_ISR_TCIF8

Channel 8 transfer complete flag

LL_DMA_ISR_HTIF8

Channel 8 half transfer flag

LL_DMA_ISR_TEIF8

Channel 8 transfer error flag

IT Defines

LL_DMA_CCR_TCIE

Transfer complete interrupt

LL_DMA_CCR_HTIE

Half Transfer interrupt

LL_DMA_CCR_TEIE

Transfer error interrupt

Memory data alignment

LL_DMA_MDATAALIGN_BYTE

Memory data alignment : Byte

LL_DMA_MDATAALIGN_HALFWORD

Memory data alignment : HalfWord

LL_DMA_MDATAALIGN_WORD

Memory data alignment : Word

Memory increment mode

LL_DMA_MEMORY_INCREMENT

Memory increment mode Enable

LL_DMA_MEMORY_NOINCREMENT

Memory increment mode Disable

Transfer mode

LL_DMA_MODE_NORMAL

Normal Mode

LL_DMA_MODE_CIRCULAR

Circular Mode

Peripheral data alignment

LL_DMA_PDATAALIGN_BYTE

Peripheral data alignment : Byte

LL_DMA_PDATAALIGN_HALFWORD

Peripheral data alignment : HalfWord

LL_DMA_PDATAALIGN_WORD

Peripheral data alignment : Word

Peripheral increment mode

LL_DMA_PERIPH_INCREMENT

Peripheral increment mode Enable

LL_DMA_PERIPH_NOINCREMENT

Peripheral increment mode Disable

Transfer Priority level

LL_DMA_PRIORITY_LOW

Priority level : Low

LL_DMA_PRIORITY_MEDIUM

Priority level : Medium

LL_DMA_PRIORITY_HIGH

Priority level : High

LL_DMA_PRIORITY_VERYHIGH

Priority level : Very_High

Convert DMAxChannely

__LL_DMA_GET_INSTANCE

Description:

- Convert DMAx_Channely into DMAx.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely

Return value:

- DMAx

__LL_DMA_GET_CHANNEL

Description:

- Convert DMAx_Channely into LL_DMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely

Return value:

- LL_DMA_CHANNEL_y

__LL_DMA_GET_CHANNEL_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_CHANNEL_y into DMAx_Channely.

Parameters:

- __DMA_INSTANCE__: DMAx
- __CHANNEL__: LL_DMA_CHANNEL_y

Return value:

- DMAx_Channely

Common Write and read registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be read

Return value:

- Register: value

75 LL EXTI Generic Driver

75.1 EXTI Firmware driver registers structures

75.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the stm32g4xx_ll_exti.h

Data Fields

- ***uint32_t Line_0_31***
- ***uint32_t Line_32_63***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_32_63***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 32 to 63 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

75.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

75.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_EnableIT_0_31

LL_EXTI_EnableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35 (*)
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42(*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- The reset value for the direct lines (lines from 32 to 34, line 39) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_EnableIT_32_63

LL_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31 (*)

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_DisableIT_0_31

LL_EXTI_DisableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35 (*)
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42(*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- The reset value for the direct lines (lines from 32 to 34, line 39) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_DisableIT_32_63

LL_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_IsEnabledIT_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35 (*)
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42(*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct lines (lines from 32 to 34, line 39) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_IsEnabledIT_32_63

LL_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31 (*)

Return values

- **None:**

Notes

- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_EnableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35 (*)
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42(*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- (*): Available in some devices

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_EnableEvent_32_63

LL_EXTI_DisableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_DisableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35 (*)
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42(*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- (*): Available in some devices

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_DisableEvent_32_63

LL_EXTI_IsEnabledEvent_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_IsEnabledEvent_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35 (*)
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42(*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- (*): Available in some devices

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_IsEnabledEvent_32_63

LL_EXTI_EnableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_EnableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_EnableRisingTrig_32_63

LL_EXTI_DisableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_DisableRisingTrig_32_63

LL_EXTI_IsEnabledRisingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **State:** of bit (1 or 0).

Notes

- (*): Available in some devices

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_IsEnabledRisingTrig_32_63

LL_EXTI_EnableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_EnableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_EnableFallingTrig_32_63

LL_EXTI_DisableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_DisableFallingTrig_32_63

LL_EXTI_IsEnabledFallingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **State:** of bit (1 or 0).

Notes

- (*): Available in some devices

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_IsEnabledFallingTrig_32_63

LL_EXTI_GenerateSWI_0_31

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR1, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR1 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR1 register (by writing a 1 into the bit)
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SWIER1 SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_GenerateSWI_32_63

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_32_63 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **None:**

Notes

- If the interrupt is enabled on this line inthe EXTI_IMR2, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR2 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR2 register (by writing a 1 into the bit)
- (*): Available in some devices

Reference Manual to LL API cross reference:

- SWIER2 SWIx LL_EXTI_GenerateSWI_32_63

LL_EXTI_IsActiveFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PR1 PIFx LL_EXTI_IsActiveFlag_0_31

LL_EXTI_IsActiveFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- PR2 PIFx LL_EXTI_IsActiveFlag_32_63

LL_EXTI_ReadFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PR1 PIFx LL_EXTI_ReadFlag_0_31

LL_EXTI_ReadFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- PR2 PIFx LL_EXTI_ReadFlag_32_63

LL_EXTI_ClearFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31 (*)

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PR1 PIFx LL_EXTI_ClearFlag_0_31

LL_EXTI_ClearFlag_32_63

Function name

__STATIC_INLINE void LL_EXTI_ClearFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32 (*)
  - LL_EXTI_LINE_33 (*)
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- (*): Available in some devices

Reference Manual to LL API cross reference:

- PR2 PIFx LL_EXTI_ClearFlag_32_63

LL_EXTI_Init

Function name

uint32_t LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.

Parameters

- **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - 0x00: EXTI registers are initialized
  - any other calue : wrong configuration

LL_EXTI_DeInit

Function name

uint32_t LL_EXTI_DeInit (void)

Function description

De-initialize the EXTI registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - 0x00: EXTI registers are de-initialized

LL_EXTI_StructInit

Function name

void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Set each LL_EXTI_InitTypeDef field to default value.

Parameters

- **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **None:**

75.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

75.3.1 EXTI

EXTI

LINE

LL_EXTI_LINE_0

Extended line 0

LL_EXTI_LINE_1

Extended line 1

LL_EXTI_LINE_2

Extended line 2

LL_EXTI_LINE_3

Extended line 3

LL_EXTI_LINE_4

Extended line 4

LL_EXTI_LINE_5

Extended line 5

LL_EXTI_LINE_6

Extended line 6

LL_EXTI_LINE_7

Extended line 7

LL_EXTI_LINE_8

Extended line 8

LL_EXTI_LINE_9

Extended line 9

LL_EXTI_LINE_10

Extended line 10

LL_EXTI_LINE_11

Extended line 11

LL_EXTI_LINE_12

Extended line 12

LL_EXTI_LINE_13

Extended line 13

LL_EXTI_LINE_14

Extended line 14

LL_EXTI_LINE_15

Extended line 15

LL_EXTI_LINE_16

Extended line 16

LL_EXTI_LINE_17

Extended line 17

LL_EXTI_LINE_18

Extended line 18

LL_EXTI_LINE_19

Extended line 19

LL_EXTI_LINE_20

Extended line 20

LL_EXTI_LINE_21

Extended line 21

LL_EXTI_LINE_22

Extended line 22

LL_EXTI_LINE_23

Extended line 23

LL_EXTI_LINE_24

Extended line 24

LL_EXTI_LINE_25

Extended line 25

LL_EXTI_LINE_26

Extended line 26

LL_EXTI_LINE_27

Extended line 27

LL_EXTI_LINE_28

Extended line 28

LL_EXTI_LINE_29

Extended line 29

LL_EXTI_LINE_30

Extended line 30

LL_EXTI_LINE_31

Extended line 31

LL_EXTI_LINE_ALL_0_31

All Extended line not reserved

LL_EXTI_LINE_32

Extended line 32

LL_EXTI_LINE_33

Extended line 33

LL_EXTI_LINE_34

Extended line 34

LL_EXTI_LINE_35

Extended line 35

LL_EXTI_LINE_36

Extended line 36

LL_EXTI_LINE_37

Extended line 37

LL_EXTI_LINE_38

Extended line 38

LL_EXTI_LINE_39

Extended line 39

LL_EXTI_LINE_40

Extended line 40

LL_EXTI_LINE_41

Extended line 41

LL_EXTI_LINE_42

Extended line 42

LL_EXTI_LINE_ALL_32_63

All Extended line not reserved

LL_EXTI_LINE_ALL

All Extended line

LL_EXTI_LINE_NONE

None Extended line

Mode

LL_EXTI_MODE_IT

Interrupt Mode

LL_EXTI_MODE_EVENT

Event Mode

LL_EXTI_MODE_IT_EVENT

Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE

No Trigger Mode

LL_EXTI_TRIGGER_RISING

Trigger Rising Mode

LL_EXTI_TRIGGER_FALLING

Trigger Falling Mode

LL_EXTI_TRIGGER_RISING_FALLING

Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

76 LL FMAC Generic Driver

76.1 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

76.1.1 Detailed description of functions

LL_FMAC_SetX1FullWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetX1FullWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure X1 full watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_SetX1FullWatermark

LL_FMAC_GetX1FullWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetX1FullWatermark (FMAC_TypeDef * FMACx)

Function description

Return X1 full watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_GetX1FullWatermark

LL_FMAC_SetX1BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX1BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X1 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_SetX1BufferSize

LL_FMAC_GetX1BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1BufferSize (FMAC_TypeDef * FMACx)

Function description

Return X1 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the input buffer (including the optional "headroom") (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_GetX1BufferSize

LL_FMAC_SetX1Base

Function name

__STATIC_INLINE void LL_FMAC_SetX1Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X1 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_SetX1Base

LL_FMAC_GetX1Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1Base (FMAC_TypeDef * FMACx)

Function description

Return X1 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the input buffer (X1) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_GetX1Base

LL_FMAC_SetX2BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX2BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X2 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_SetX2BufferSize

LL_FMAC_GetX2BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2BufferSize (FMAC_TypeDef * FMACx)

Function description

Return X2 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the coefficient buffer (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_GetX2BufferSize

LL_FMAC_SetX2Base

Function name

__STATIC_INLINE void LL_FMAC_SetX2Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X2 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_SetX2Base

LL_FMAC_GetX2Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2Base (FMAC_TypeDef * FMACx)

Function description

Return X2 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the coefficient buffer (X2) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_GetX2Base

LL_FMAC_SetYEmptyWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetYEmptyWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure Y empty watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_SetYEmptyWatermark

LL_FMAC_GetYEmptyWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetYEmptyWatermark (FMAC_TypeDef * FMACx)

Function description

Return Y empty watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_GetYEmptyWatermark

LL_FMAC_SetYBufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetYBufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure Y buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_SetYBufferSize

LL_FMAC_GetYBufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBufferSize (FMAC_TypeDef * FMACx)

Function description

Return Y buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the output buffer (including the optional "headroom" - value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_GetYBufferSize

LL_FMAC_SetYBase

Function name

__STATIC_INLINE void LL_FMAC_SetYBase (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure Y base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_SetYBase

LL_FMAC_GetYBase

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBase (FMAC_TypeDef * FMACx)

Function description

Return Y base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the output buffer (Y) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_GetYBase

LL_FMAC_EnableStart

Function name

__STATIC_INLINE void LL_FMAC_EnableStart (FMAC_TypeDef * FMACx)

Function description

Start FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_EnableStart

LL_FMAC_DisableStart

Function name

__STATIC_INLINE void LL_FMAC_DisableStart (FMAC_TypeDef * FMACx)

Function description

Stop FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_DisableStart

LL_FMAC_IsEnabledStart

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledStart (FMAC_TypeDef * FMACx)

Function description

Check the state of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_IsEnabledStart

LL_FMAC_SetFunction

Function name

__STATIC_INLINE void LL_FMAC_SetFunction (FMAC_TypeDef * FMACx, uint32_t Function)

Function description

Configure function.

Parameters

- **FMACx:** FMAC instance
- **Function:** This parameter can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_SetFunction

LL_FMAC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetFunction (FMAC_TypeDef * FMACx)

Function description

Return function.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_GetFunction

LL_FMAC_SetParamR

Function name

__STATIC_INLINE void LL_FMAC_SetParamR (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter R.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_SetParamR

LL_FMAC_GetParamR

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamR (FMAC_TypeDef * FMACx)

Function description

Return input parameter R.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter R (gain, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_GetParamR

LL_FMAC_SetParamQ

Function name

__STATIC_INLINE void LL_FMAC_SetParamQ (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter Q.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_SetParamQ

LL_FMAC_GetParamQ

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamQ (FMAC_TypeDef * FMACx)

Function description

Return input parameter Q.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter Q (vector length, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_GetParamQ

LL_FMAC_SetParamP

Function name

__STATIC_INLINE void LL_FMAC_SetParamP (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter P.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_SetParamP

LL_FMAC_GetParamP

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamP (FMAC_TypeDef * FMACx)

Function description

Return input parameter P.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter P (vector length, number of filter taps, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_GetParamP

LL_FMAC_EnableReset

Function name

__STATIC_INLINE void LL_FMAC_EnableReset (FMAC_TypeDef * FMACx)

Function description

Start the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_EnableReset

LL_FMAC_IsEnabledReset

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledReset (FMAC_TypeDef * FMACx)

Function description

Check the state of the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_IsEnabledReset

LL_FMAC_EnableClipping

Function name

__STATIC_INLINE void LL_FMAC_EnableClipping (FMAC_TypeDef * FMACx)

Function description

Enable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_EnableClipping

LL_FMAC_DisableClipping

Function name

__STATIC_INLINE void LL_FMAC_DisableClipping (FMAC_TypeDef * FMACx)

Function description

Disable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_DisableClipping

LL_FMAC_IsEnabledClipping

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledClipping (FMAC_TypeDef * FMACx)

Function description

Check Clipping State.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_IsEnabledClipping

LL_FMAC_EnableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_EnableDMAReq_WRITE

LL_FMAC_DisableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_DisableDMAReq_WRITE

LL_FMAC_IsEnabledDMAReq_WRITE

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA write channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_IsEnabledDMAReq_WRITE

LL_FMAC_EnableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_EnableDMAReq_READ

LL_FMAC_DisableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_DisableDMAReq_READ

LL_FMAC_IsEnabledDMAReq_READ

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA read channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_IsEnabledDMAReq_READ

LL_FMAC_EnableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Enable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_EnableIT_SAT

LL_FMAC_DisableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Disable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_DisableIT_SAT

LL_FMAC_IsEnabledIT_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_SAT (FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_IsEnabledIT_SAT

LL_FMAC_EnableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_EnableIT_UNFL

LL_FMAC_DisableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_DisableIT_UNFL

LL_FMAC_IsEnabledIT_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_IsEnabledIT_UNFL

LL_FMAC_EnableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_EnableIT_OVFL

LL_FMAC_DisableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_DisableIT_OVFL

LL_FMAC_IsEnabledIT_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_IsEnabledIT_OVFL

LL_FMAC_EnableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_WR (FMAC_TypeDef * FMACx)

Function description

Enable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_EnableIT_WR

LL_FMAC_DisableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_WR (FMAC_TypeDef * FMACx)

Function description

Disable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_DisableIT_WR

LL_FMAC_IsEnabledIT_WR

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_WR (FMAC_TypeDef * FMACx)

Function description

Check FMAC write interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_IsEnabledIT_WR

LL_FMAC_EnableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_RD (FMAC_TypeDef * FMACx)

Function description

Enable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_EnableIT_RD

LL_FMAC_DisableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_RD (FMAC_TypeDef * FMACx)

Function description

Disable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_DisableIT_RD

LL_FMAC_IsEnabledIT_RD

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_RD (FMAC_TypeDef * FMACx)

Function description

Check FMAC read interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_IsEnabledIT_RD

LL_FMAC_IsActiveFlag_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_SAT (FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SAT LL_FMAC_IsActiveFlag_SAT

LL_FMAC_IsActiveFlag_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_UNFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UNFL LL_FMAC_IsActiveFlag_UNFL

LL_FMAC_IsActiveFlag_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_OVFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVFL LL_FMAC_IsActiveFlag_OVFL

LL_FMAC_IsActiveFlag_X1FULL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_X1FULL (FMAC_TypeDef * FMACx)

Function description

Check FMAC X1 buffer full flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR X1FULL LL_FMAC_IsActiveFlag_X1FULL

LL_FMAC_IsActiveFlag_YEMPTY

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_YEMPTY (FMAC_TypeDef * FMACx)

Function description

Check FMAC Y buffer empty flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR YEMPTY LL_FMAC_IsActiveFlag_YEMPTY

LL_FMAC_WriteData

Function name

__STATIC_INLINE void LL_FMAC_WriteData (FMAC_TypeDef * FMACx, uint16_t InData)

Function description

Write 16-bit input data for the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **InData:** 16-bit value to be provided as input data for FMAC processing. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA WDATA LL_FMAC_WriteData

LL_FMAC_ReadData

Function name

__STATIC_INLINE uint16_t LL_FMAC_ReadData (FMAC_TypeDef * FMACx)

Function description

Return 16-bit output data of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint16_t:** 16-bit output data of FMAC processing (value between Min_Data=0x0000 and Max_Data=0xFFFF).

Reference Manual to LL API cross reference:

- RDATA RDATA LL_FMAC_ReadData

LL_FMAC_ConfigX1

Function name

__STATIC_INLINE void LL_FMAC_ConfigX1 (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X1 buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_ConfigX1
- X1BUFCFG X1_BASE LL_FMAC_ConfigX1
- X1BUFCFG X1_BUF_SIZE LL_FMAC_ConfigX1

LL_FMAC_ConfigX2

Function name

__STATIC_INLINE void LL_FMAC_ConfigX2 (FMAC_TypeDef * FMACx, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X2 buffer.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_ConfigX2
- X2BUFCFG X2_BUF_SIZE LL_FMAC_ConfigX2

LL_FMAC_ConfigY

Function name

__STATIC_INLINE void LL_FMAC_ConfigY (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for Y buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_ConfigY
- YBUFCFG Y_BASE LL_FMAC_ConfigY
- YBUFCFG Y_BUF_SIZE LL_FMAC_ConfigY

LL_FMAC_ConfigFunc

Function name

__STATIC_INLINE void LL_FMAC_ConfigFunc (FMAC_TypeDef * FMACx, uint8_t Start, uint32_t Function, uint8_t ParamP, uint8_t ParamQ, uint8_t ParamR)

Function description

Configure the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **Start:** This parameter can be one of the following values:
  - LL_FMAC_PROCESSING_STOP
  - LL_FMAC_PROCESSING_START
- **Function:** This parameter can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1
- **ParamP:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamQ:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamR:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_ConfigFunc
- PARAM FUNC LL_FMAC_ConfigFunc
- PARAM P LL_FMAC_ConfigFunc
- PARAM Q LL_FMAC_ConfigFunc
- PARAM R LL_FMAC_ConfigFunc

LL_FMAC_Init

Function name

ErrorStatus LL_FMAC_Init (FMAC_TypeDef * FMACx)

Function description

Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **ErrorStatus:** enumeration value:
  - SUCCESS: FMAC registers are initialized
  - ERROR: FMAC registers are not initialized

LL_FMAC_DeInit

Function name

ErrorStatus LL_FMAC_DeInit (FMAC_TypeDef * FMACx)

Function description

De-Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: FMAC registers are de-initialized
  - ERROR: FMAC registers are not de-initialized

76.2 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

76.2.1 FMAC

FMAC

FMAC functions

LL_FMAC_FUNC_LOAD_X1

Load X1 buffer

LL_FMAC_FUNC_LOAD_X2

Load X2 buffer

LL_FMAC_FUNC_LOAD_Y

Load Y buffer

LL_FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

LL_FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

Get Flag Defines

LL_FMAC_SR_SAT

Saturation Error Flag (this helps in debugging a filter)

LL_FMAC_SR_UNFL

Underflow Error Flag

LL_FMAC_SR_OVFL

Overflow Error Flag

LL_FMAC_SR_X1FULL

X1 Buffer Full Flag

LL_FMAC_SR_YEMPTY

Y Buffer Empty Flag

IT Defines

LL_FMAC_CR_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

LL_FMAC_CR_UNFLIEN

Underflow Error Interrupt Enable

LL_FMAC_CR_OVFLIEN

Overflow Error Interrupt Enable

LL_FMAC_CR_WIEN

Write Interrupt Enable

LL_FMAC_CR_RIEN

Read Interrupt Enable

FMAC processing

LL_FMAC_PROCESSING_STOP

Stop FMAC Processing

LL_FMAC_PROCESSING_START

Start FMAC Processing

FMAC watermarks

LL_FMAC_WM_0_THRESHOLD_1

Buffer full/empty flag set if there is less than 1 free/unread space.

LL_FMAC_WM_1_THRESHOLD_2

Buffer full/empty flag set if there are less than 2 free/unread spaces.

LL_FMAC_WM_2_THRESHOLD_4

Buffer full/empty flag set if there are less than 4 free/unread spaces.

LL_FMAC_WM_3_THRESHOLD_8

Buffer full/empty flag set if there are less than 8 free/empty spaces.

Common Write and read registers Macros

LL_FMAC_WriteReg

Description:

- Write a value in FMAC register.

Parameters:

- __INSTANCE__: FMAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_FMAC_ReadReg

Description:

- Read a value in FMAC register.

Parameters:

- __INSTANCE__: FMAC Instance
- __REG__: Register to be read

Return value:

- Register: value

77 LL GPIO Generic Driver

77.1 GPIO Firmware driver registers structures

77.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32g4xx_ll_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***

  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***

  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***

  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***

  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***

  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.
- ***uint32_t LL_GPIO_InitTypeDef::Alternate***

  Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_LL_EC_AF***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetAFPin_0_7()** and **LL_GPIO_SetAFPin_8_15()**.

77.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

77.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name

__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)

Function description

Configure gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Return values

- **None:**

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name

__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)

Function description

Configure gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL
- **OutputType:** This parameter can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Return values

- **None:**

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_SetPinOutputType

LL_GPIO_GetPinOutputType

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name

__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)

Function description

Configure gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Return values

- **None:**

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name

__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)

Function description

Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Pull:** This parameter can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Notes

- Possible values are from AF0 to AF15 depending on target.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name

__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a specific configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return 1 if all pins passed as parameter, of a dedicated port, are locked.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (GPIO_TypeDef * GPIOx)

Function description

Return 1 if one of the pin of a dedicated port is locked.

Parameters

- **GPIOx:** GPIO Port

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (GPIO_TypeDef * GPIOx)

Function description

Return full input data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Input:** data register value of port

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name

__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)

Function description

Write output data register for the port.

Parameters

- **GPIOx:** GPIO Port
- **PortValue:** Level value for each pin of the port

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)

Function description

Return full output data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Output:** data register value of port

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to low level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name

__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Toggle data value for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_TogglePin

LL_GPIO_DeInit

Function name

ErrorStatus LL_GPIO_DeInit (GPIO_TypeDef * GPIOx)

Function description

De-initialize GPIO registers (Registers restored to their default values).

Parameters

- **GPIOx:** GPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are de-initialized
  - ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name

ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content
  - ERROR: Not applicable

LL_GPIO_StructInit

Function name

void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Set each LL_GPIO_InitTypeDef field to default value.

Parameters

- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

77.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

77.3.1 GPIO

GPIO

Alternate Function

LL_GPIO_AF_0

Select alternate function 0

LL_GPIO_AF_1

Select alternate function 1

LL_GPIO_AF_2

Select alternate function 2

LL_GPIO_AF_3

Select alternate function 3

LL_GPIO_AF_4

Select alternate function 4

LL_GPIO_AF_5

Select alternate function 5

LL_GPIO_AF_6

Select alternate function 6

LL_GPIO_AF_7

Select alternate function 7

LL_GPIO_AF_8

Select alternate function 8

LL_GPIO_AF_9

Select alternate function 9

LL_GPIO_AF_10

Select alternate function 10

LL_GPIO_AF_11

Select alternate function 11

LL_GPIO_AF_12

Select alternate function 12

LL_GPIO_AF_13

Select alternate function 13

LL_GPIO_AF_14

Select alternate function 14

LL_GPIO_AF_15

Select alternate function 15

Mode

LL_GPIO_MODE_INPUT

Select input mode

LL_GPIO_MODE_OUTPUT

Select output mode

LL_GPIO_MODE_ALTERNATE

Select alternate function mode

LL_GPIO_MODE_ANALOG

Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL

Select push-pull as output type

LL_GPIO_OUTPUT_OPENDRAIN

Select open-drain as output type

PIN

LL_GPIO_PIN_0

Select pin 0

LL_GPIO_PIN_1

Select pin 1

LL_GPIO_PIN_2

Select pin 2

LL_GPIO_PIN_3

Select pin 3

LL_GPIO_PIN_4

Select pin 4

LL_GPIO_PIN_5

Select pin 5

LL_GPIO_PIN_6

Select pin 6

LL_GPIO_PIN_7

Select pin 7

LL_GPIO_PIN_8

Select pin 8

LL_GPIO_PIN_9

Select pin 9

LL_GPIO_PIN_10

Select pin 10

LL_GPIO_PIN_11

Select pin 11

LL_GPIO_PIN_12

Select pin 12

LL_GPIO_PIN_13

Select pin 13

LL_GPIO_PIN_14

Select pin 14

LL_GPIO_PIN_15

Select pin 15

LL_GPIO_PIN_ALL

Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO

Select I/O no pull

LL_GPIO_PULL_UP

Select I/O pull up

LL_GPIO_PULL_DOWN

Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW

Select I/O low output speed

LL_GPIO_SPEED_FREQ_MEDIUM

Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH

Select I/O fast output speed

LL_GPIO_SPEED_FREQ_VERY_HIGH

Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

78 LL HRTIM Generic Driver

78.1 HRTIM Firmware driver API description

The following section lists the various functions of the HRTIM library.

78.1.1 Detailed description of functions

LL_HRTIM_SetSyncInSrc

Function name

__STATIC_INLINE void LL_HRTIM_SetSyncInSrc (HRTIM_TypeDef * HRTIMx, uint32_t SyncInSrc)

Function description

Select the HRTIM synchronization input source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **SyncInSrc:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCIN_SRC_NONE
  - LL_HRTIM_SYNCIN_SRC_TIM_EVENT
  - LL_HRTIM_SYNCIN_SRC_EXTERNAL_EVENT

Return values

- **None:**

Notes

- This function must not be called when the concerned timer(s) is (are) enabled .

Reference Manual to LL API cross reference:

- MCR SYNCIN LL_HRTIM_SetSyncInSrc

LL_HRTIM_GetSyncInSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetSyncInSrc (HRTIM_TypeDef * HRTIMx)

Function description

Get actual HRTIM synchronization input source.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **SyncInSrc:** Returned value can be one of the following values:
  - LL_HRTIM_SYNCIN_SRC_NONE
  - LL_HRTIM_SYNCIN_SRC_TIM_EVENT
  - LL_HRTIM_SYNCIN_SRC_EXTERNAL_EVENT

Reference Manual to LL API cross reference:

- MCR SYNCIN LL_HRTIM_SetSyncInSrc

LL_HRTIM_ConfigSyncOut

Function name

__STATIC_INLINE void LL_HRTIM_ConfigSyncOut (HRTIM_TypeDef * HRTIMx, uint32_t Config, uint32_t Src)

Function description

Configure the HRTIM synchronization output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Config:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_DISABLED
  - LL_HRTIM_SYNCOUT_POSITIVE_PULSE
  - LL_HRTIM_SYNCOUT_NEGATIVE_PULSE
- **Src:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_SRC_MASTER_START
  - LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1
  - LL_HRTIM_SYNCOUT_SRC_TIMA_START
  - LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSRC LL_HRTIM_ConfigSyncOut
- MCR SYNCOUT LL_HRTIM_ConfigSyncOut

LL_HRTIM_SetSyncOutConfig

Function name

__STATIC_INLINE void LL_HRTIM_SetSyncOutConfig (HRTIM_TypeDef * HRTIMx, uint32_t SyncOutConfig)

Function description

Set the routing and conditioning of the synchronization output event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **SyncOutConfig:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_DISABLED
  - LL_HRTIM_SYNCOUT_POSITIVE_PULSE
  - LL_HRTIM_SYNCOUT_NEGATIVE_PULSE

Return values

- **None:**

Notes

- This function can be called only when the master timer is enabled.

Reference Manual to LL API cross reference:

- MCR SYNCOUT LL_HRTIM_SetSyncOutConfig

LL_HRTIM_GetSyncOutConfig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetSyncOutConfig (HRTIM_TypeDef * HRTIMx)

Function description

Get actual routing and conditioning of the synchronization output event.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **SyncOutConfig:** Returned value can be one of the following values:
  - LL_HRTIM_SYNCOUT_DISABLED
  - LL_HRTIM_SYNCOUT_POSITIVE_PULSE
  - LL_HRTIM_SYNCOUT_NEGATIVE_PULSE

Reference Manual to LL API cross reference:

- MCR SYNCOUT LL_HRTIM_GetSyncOutConfig

LL_HRTIM_SetSyncOutSrc

Function name

__STATIC_INLINE void LL_HRTIM_SetSyncOutSrc (HRTIM_TypeDef * HRTIMx, uint32_t SyncOutSrc)

Function description

Set the source and event to be sent on the HRTIM synchronization output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **SyncOutSrc:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_SRC_MASTER_START
  - LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1
  - LL_HRTIM_SYNCOUT_SRC_TIMA_START
  - LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSRC LL_HRTIM_SetSyncOutSrc

LL_HRTIM_GetSyncOutSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetSyncOutSrc (HRTIM_TypeDef * HRTIMx)

Function description

Get actual source and event sent on the HRTIM synchronization output.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **SyncOutSrc:** Returned value can be one of the following values:
  - LL_HRTIM_SYNCOUT_SRC_MASTER_START
  - LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1
  - LL_HRTIM_SYNCOUT_SRC_TIMA_START
  - LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

Reference Manual to LL API cross reference:

- MCR SYNCSRC LL_HRTIM_GetSyncOutSrc

LL_HRTIM_SuspendUpdate

Function name

__STATIC_INLINE void LL_HRTIM_SuspendUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Disable (temporarily) update event generation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- Allow to temporarily disable the transfer from preload to active registers, whatever the selected update event. This allows to modify several registers in multiple timers.

Reference Manual to LL API cross reference:

- CR1 MUDIS LL_HRTIM_SuspendUpdate
- CR1 TAUDIS LL_HRTIM_SuspendUpdate
- CR1 TBUDIS LL_HRTIM_SuspendUpdate
- CR1 TCUDIS LL_HRTIM_SuspendUpdate
- CR1 TDUDIS LL_HRTIM_SuspendUpdate
- CR1 TEUDIS LL_HRTIM_SuspendUpdate
- CR1 TFUDIS LL_HRTIM_SuspendUpdate

LL_HRTIM_ResumeUpdate

Function name

__STATIC_INLINE void LL_HRTIM_ResumeUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Enable update event generation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- The regular update event takes place.

Reference Manual to LL API cross reference:

- CR1 MUDIS LL_HRTIM_ResumeUpdate
- CR1 TAUDIS LL_HRTIM_ResumeUpdate
- CR1 TBUDIS LL_HRTIM_ResumeUpdate
- CR1 TCUDIS LL_HRTIM_ResumeUpdate
- CR1 TDUDIS LL_HRTIM_ResumeUpdate
- CR1 TEUDIS LL_HRTIM_ResumeUpdate
- CR1 TFUDIS LL_HRTIM_ResumeUpdate

LL_HRTIM_ForceUpdate

Function name

__STATIC_INLINE void LL_HRTIM_ForceUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Force an immediate transfer from the preload to the active register .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- Any pending update request is cancelled.

Reference Manual to LL API cross reference:

- CR2 MSWU LL_HRTIM_ForceUpdate
- CR2 TASWU LL_HRTIM_ForceUpdate
- CR2 TBSWU LL_HRTIM_ForceUpdate
- CR2 TCSWU LL_HRTIM_ForceUpdate
- CR2 TDSWU LL_HRTIM_ForceUpdate
- CR2 TESWU LL_HRTIM_ForceUpdate
- CR2 TFSWU LL_HRTIM_ForceUpdate

LL_HRTIM_CounterReset

Function name

__STATIC_INLINE void LL_HRTIM_CounterReset (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Reset the HRTIM timer(s) counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 MRST LL_HRTIM_CounterReset
- CR2 TARST LL_HRTIM_CounterReset
- CR2 TBRST LL_HRTIM_CounterReset
- CR2 TCRST LL_HRTIM_CounterReset
- CR2 TDRST LL_HRTIM_CounterReset
- CR2 TERST LL_HRTIM_CounterReset
- CR2 TFRST LL_HRTIM_CounterReset

LL_HRTIM_EnableSwapOutputs

Function name

__STATIC_INLINE void LL_HRTIM_EnableSwapOutputs (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

enable the swap of the Timer Output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- the HRTIM_SETA1R and HRTIM_RSTA1R are coding for the output A2, and the HRTIM_SETA2R and HRTIM_RSTA2R are coding for the output A1
- This bit is not significant when the Push-pull mode is enabled (PSHPLL = 1)

Reference Manual to LL API cross reference:

- CR2 SWPA LL_HRTIM_EnableSwapOutputs
- CR2 SWPB LL_HRTIM_EnableSwapOutputs
- CR2 SWPC LL_HRTIM_EnableSwapOutputs
- CR2 SWPD LL_HRTIM_EnableSwapOutputs
- CR2 SWPE LL_HRTIM_EnableSwapOutputs
- CR2 SWPF LL_HRTIM_EnableSwapOutputs

LL_HRTIM_DisableSwapOutputs

Function name

__STATIC_INLINE void LL_HRTIM_DisableSwapOutputs (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

disable the swap of the Timer Output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- the HRTIM_SETA1R and HRTIM_RSTA1R are coding for the output A1, and the HRTIM_SETA2R and HRTIM_RSTA2R are coding for the output A2
- This bit is not significant when the Push-pull mode is enabled (PSHPLL = 1)

Reference Manual to LL API cross reference:

- CR2 SWPA LL_HRTIM_DisableSwapOutputs
- CR2 SWPB LL_HRTIM_DisableSwapOutputs
- CR2 SWPC LL_HRTIM_DisableSwapOutputs
- CR2 SWPD LL_HRTIM_DisableSwapOutputs
- CR2 SWPE LL_HRTIM_DisableSwapOutputs
- CR2 SWPF LL_HRTIM_DisableSwapOutputs

LL_HRTIM_IsEnabledSwapOutputs

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledSwapOutputs (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

reports the Timer Outputs swap position.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **1:** HRTIM_SETA1R and HRTIM_RSTA1R are coding for the output A2, HRTIM_SETA2R and HRTIM_RSTA2R are coding for the output A1 0: HRTIM_SETA1R and HRTIM_RSTA1R are coding for the output A1, HRTIM_SETA2R and HRTIM_RSTA2R are coding for the output A2

Notes

- This bit is not significant when the Push-pull mode is enabled (PSHPLL = 1)

Reference Manual to LL API cross reference:

- CR2 SWPA LL_HRTIM_IsEnabledSwapOutputs
- CR2 SWPB LL_HRTIM_IsEnabledSwapOutputs
- CR2 SWPC LL_HRTIM_IsEnabledSwapOutputs
- CR2 SWPD LL_HRTIM_IsEnabledSwapOutputs
- CR2 SWPE LL_HRTIM_IsEnabledSwapOutputs
- CR2 SWPF LL_HRTIM_IsEnabledSwapOutputs

LL_HRTIM_EnableOutput

Function name

__STATIC_INLINE void LL_HRTIM_EnableOutput (HRTIM_TypeDef * HRTIMx, uint32_t Outputs)

Function description

Enable the HRTIM timer(s) output(s) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Outputs:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **None:**

Reference Manual to LL API cross reference:

- OENR TA1OEN LL_HRTIM_EnableOutput
- OENR TA2OEN LL_HRTIM_EnableOutput
- OENR TB1OEN LL_HRTIM_EnableOutput
- OENR TB2OEN LL_HRTIM_EnableOutput
- OENR TC1OEN LL_HRTIM_EnableOutput
- OENR TC2OEN LL_HRTIM_EnableOutput
- OENR TD1OEN LL_HRTIM_EnableOutput
- OENR TD2OEN LL_HRTIM_EnableOutput
- OENR TE1OEN LL_HRTIM_EnableOutput
- OENR TE2OEN LL_HRTIM_EnableOutput
- OENR TF1OEN LL_HRTIM_EnableOutput
- OENR TF2OEN LL_HRTIM_EnableOutput

LL_HRTIM_DisableOutput

Function name

__STATIC_INLINE void LL_HRTIM_DisableOutput (HRTIM_TypeDef * HRTIMx, uint32_t Outputs)

Function description

Disable the HRTIM timer(s) output(s) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Outputs:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **None:**

Reference Manual to LL API cross reference:

- OENR TA1OEN LL_HRTIM_DisableOutput
- OENR TA2OEN LL_HRTIM_DisableOutput
- OENR TB1OEN LL_HRTIM_DisableOutput
- OENR TB2OEN LL_HRTIM_DisableOutput
- OENR TC1OEN LL_HRTIM_DisableOutput
- OENR TC2OEN LL_HRTIM_DisableOutput
- OENR TD1OEN LL_HRTIM_DisableOutput
- OENR TD2OEN LL_HRTIM_DisableOutput
- OENR TE1OEN LL_HRTIM_DisableOutput
- OENR TE2OEN LL_HRTIM_DisableOutput
- OENR TF1OEN LL_HRTIM_DisableOutput
- OENR TF2OEN LL_HRTIM_DisableOutput

LL_HRTIM_IsEnabledOutput

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledOutput (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Indicates whether the HRTIM timer output is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **State:** of TxyOEN bit in HRTIM_OENR register (1 or 0).

Reference Manual to LL API cross reference:

- OENR TA1OEN LL_HRTIM_IsEnabledOutput
- OENR TA2OEN LL_HRTIM_IsEnabledOutput
- OENR TB1OEN LL_HRTIM_IsEnabledOutput
- OENR TB2OEN LL_HRTIM_IsEnabledOutput
- OENR TC1OEN LL_HRTIM_IsEnabledOutput
- OENR TC2OEN LL_HRTIM_IsEnabledOutput
- OENR TD1OEN LL_HRTIM_IsEnabledOutput
- OENR TD2OEN LL_HRTIM_IsEnabledOutput
- OENR TE1OEN LL_HRTIM_IsEnabledOutput
- OENR TE2OEN LL_HRTIM_IsEnabledOutput
- OENR TF1OEN LL_HRTIM_IsEnabledOutput
- OENR TF2OEN LL_HRTIM_IsEnabledOutput

LL_HRTIM_IsDisabledOutput

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsDisabledOutput (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Indicates whether the HRTIM timer output is disabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **State:** of TxyODS bit in HRTIM_OENR register (1 or 0).

Reference Manual to LL API cross reference:

- ODISR TA1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TA2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TB1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TB2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TC1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TC2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TD1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TD2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TE1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TE2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TF1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TF2ODIS LL_HRTIM_IsDisabledOutput

LL_HRTIM_ConfigADCTrig

Function name

__STATIC_INLINE void LL_HRTIM_ConfigADCTrig (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t Update, uint32_t Src)

Function description

Configure an ADC trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
- **Update:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_UPDATE_MASTER
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_A
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_B
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_C
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_D
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_E
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_F
- **Src:** This parameter can be a combination of the following values:

Reference Manual to LL API cross reference:

- CR1 ADC1USRC LL_HRTIM_ConfigADCTrig
- CR1 ADC2USRC LL_HRTIM_ConfigADCTrig
- CR1 ADC3USRC LL_HRTIM_ConfigADCTrig
- CR1 ADC4USRC LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC1 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV1 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV5 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TFC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TARST LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TFC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBRST LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TFC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TFPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TFRST LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC1 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV6 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV7 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV8 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV9 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV10 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TAC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TFC2 LL_HRTIM_ConfigADCTrig

- ADC2R ADC2TAC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TAPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TFC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TFC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCRST LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TFPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDRST LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TEC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TEC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TEC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TERST LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC1 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV1 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV5 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TFC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TARST LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TFC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBRST LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TFC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TFPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TFRST LL_HRTIM_ConfigADCTrig

- ADC3R ADC3TEC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TEC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TEPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC1 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV6 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV7 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV8 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV9 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV10 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TFC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TFC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TFC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCRST LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TFPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDRST LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TEC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TEC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TEC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TERST LL_HRTIM_ConfigADCTrig

LL_HRTIM_SetADCTrigUpdate

Function name

__STATIC_INLINE void LL_HRTIM_SetADCTrigUpdate (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t Update)

Function description

Associate the ADCx trigger to a timer triggering the update of the HRTIM_ADCxR register.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
  - LL_HRTIM_ADCTRIG_5
  - LL_HRTIM_ADCTRIG_6
  - LL_HRTIM_ADCTRIG_7
  - LL_HRTIM_ADCTRIG_8
  - LL_HRTIM_ADCTRIG_9
  - LL_HRTIM_ADCTRIG_10
- **Update:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_UPDATE_MASTER
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_A
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_B
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_C
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_D
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_E
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_F

Return values

- **None:**

Notes

- When the preload is disabled in the source timer, the HRTIM_ADCxR registers are not preloaded either: a write access will result in an immediate update of the trigger source.

Reference Manual to LL API cross reference:

- CR1 ADC1USRC LL_HRTIM_SetADCTrigUpdate
- CR1 ADC2USRC LL_HRTIM_SetADCTrigUpdate
- CR1 ADC3USRC LL_HRTIM_SetADCTrigUpdate
- CR1 ADC4USRC LL_HRTIM_SetADCTrigUpdate
- ADCUR ADC5USRC LL_HRTIM_SetADCTrigUpdate
- ADCUR ADC6USRC LL_HRTIM_SetADCTrigUpdate
- ADCUR ADC7USRC LL_HRTIM_SetADCTrigUpdate
- ADCUR ADC8USRC LL_HRTIM_SetADCTrigUpdate
- ADCUR ADC9USRC LL_HRTIM_SetADCTrigUpdate
- ADCUR ADC10USRC LL_HRTIM_SetADCTrigUpdate

LL_HRTIM_GetADCTrigUpdate

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetADCTrigUpdate (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig)

Function description

Get the source timer triggering the update of the HRTIM_ADCxR register.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
  - LL_HRTIM_ADCTRIG_5
  - LL_HRTIM_ADCTRIG_6
  - LL_HRTIM_ADCTRIG_7
  - LL_HRTIM_ADCTRIG_8
  - LL_HRTIM_ADCTRIG_9
  - LL_HRTIM_ADCTRIG_10

Return values

- **Update:** Returned value can be one of the following values:
  - LL_HRTIM_ADCTRIG_UPDATE_MASTER
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_A
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_B
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_C
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_D
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_E
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_F

Reference Manual to LL API cross reference:

- CR1 ADC1USRC LL_HRTIM_GetADCTrigUpdate
- CR1 ADC2USRC LL_HRTIM_GetADCTrigUpdate
- CR1 ADC3USRC LL_HRTIM_GetADCTrigUpdate
- CR1 ADC4USRC LL_HRTIM_GetADCTrigUpdate
- ADCUR ADC5USRC LL_HRTIM_GetADCTrigUpdate
- ADCUR ADC6USRC LL_HRTIM_GetADCTrigUpdate
- ADCUR ADC7USRC LL_HRTIM_GetADCTrigUpdate
- ADCUR ADC8USRC LL_HRTIM_GetADCTrigUpdate
- ADCUR ADC9USRC LL_HRTIM_GetADCTrigUpdate
- ADCUR ADC10USRC LL_HRTIM_GetADCTrigUpdate

LL_HRTIM_SetADCTrigSrc

Function name

__STATIC_INLINE void LL_HRTIM_SetADCTrigSrc (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t Src)

Function description

Specify which events (timer events and/or external events) are used as triggers for ADC conversion.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
  - LL_HRTIM_ADCTRIG_5
  - LL_HRTIM_ADCTRIG_6
  - LL_HRTIM_ADCTRIG_7
  - LL_HRTIM_ADCTRIG_8
  - LL_HRTIM_ADCTRIG_9
  - LL_HRTIM_ADCTRIG_10

- **Src:** For ADC trigger 1 and ADC trigger 3 this parameter can be a combination of the following values:
  - LL_HRTIM_ADCTRIG_SRC13_NONE
  - LL_HRTIM_ADCTRIG_SRC13_MCMP1
  - LL_HRTIM_ADCTRIG_SRC13_MCMP2
  - LL_HRTIM_ADCTRIG_SRC13_MCMP3
  - LL_HRTIM_ADCTRIG_SRC13_MCMP4
  - LL_HRTIM_ADCTRIG_SRC13_MPER
  - LL_HRTIM_ADCTRIG_SRC13_EEV1
  - LL_HRTIM_ADCTRIG_SRC13_EEV2
  - LL_HRTIM_ADCTRIG_SRC13_EEV3
  - LL_HRTIM_ADCTRIG_SRC13_EEV4
  - LL_HRTIM_ADCTRIG_SRC13_EEV5
  - LL_HRTIM_ADCTRIG_SRC13_TIMACMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMACMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMAPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMARST
  - LL_HRTIM_ADCTRIG_SRC13_TIMBCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMBCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMBPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMBRST
  - LL_HRTIM_ADCTRIG_SRC13_TIMCCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMCCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMCPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMDCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMDCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMDPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMECMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMECMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMEPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMFCMP2
  - LL_HRTIM_ADCTRIG_SRC13_TIMFCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMFCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMFPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMFRST

  For ADC trigger 2 and ADC trigger 4 this parameter can be a combination of the following values:
  - LL_HRTIM_ADCTRIG_SRC24_NONE
  - LL_HRTIM_ADCTRIG_SRC24_MCMP1
  - LL_HRTIM_ADCTRIG_SRC24_MCMP2
  - LL_HRTIM_ADCTRIG_SRC24_MCMP3
  - LL_HRTIM_ADCTRIG_SRC24_MCMP4
  - LL_HRTIM_ADCTRIG_SRC24_MPER
  - LL_HRTIM_ADCTRIG_SRC24_EEV6
  - LL_HRTIM_ADCTRIG_SRC24_EEV7
  - LL_HRTIM_ADCTRIG_SRC24_EEV8
  - LL_HRTIM_ADCTRIG_SRC24_EEV9
  - LL_HRTIM_ADCTRIG_SRC24_EEV10
  - LL_HRTIM_ADCTRIG_SRC24_TIMACMP2
  - LL_HRTIM_ADCTRIG_SRC24_TIMACMP4
  - LL_HRTIM_ADCTRIG_SRC24_TIMAPER
  - LL_HRTIM_ADCTRIG_SRC24_TIMBCMP2
  - LL_HRTIM_ADCTRIG_SRC24_TIMBCMP4
  - LL_HRTIM_ADCTRIG_SRC24_TIMBPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- ADC1R ADC1MC1 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV1 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV5 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TFC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TARST LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TFC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBRST LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TFC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TFPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TFRST LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC1 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV6 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV7 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV8 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV9 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV10 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TFC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TBC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TFC3 LL_HRTIM_SetADCTrigSrc

- ADC2R ADC2TBC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TBPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TFC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCRST LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TFPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDRST LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TEC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TEC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TEC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TERST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC1 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV1 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV5 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TFC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TARST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TFC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBRST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TFC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TFPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TFRST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MC1 LL_HRTIM_SetADCTrigSrc

- ADC4R ADC4MC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV6 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV7 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV8 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV9 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV10 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TFC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TFC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TFC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCRST LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TFPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDRST LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TEC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TEC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TEC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TERST LL_HRTIM_SetADCTrigSrc
- ADCER ADC5TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC6TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC7TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC8TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC9TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC10TRG LL_HRTIM_SetADCTrigSrc

LL_HRTIM_GetADCTrigSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetADCTrigSrc (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig)

Function description

Indicate which events (timer events and/or external events) are currently used as triggers for ADC conversion.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
  - LL_HRTIM_ADCTRIG_5
  - LL_HRTIM_ADCTRIG_6
  - LL_HRTIM_ADCTRIG_7
  - LL_HRTIM_ADCTRIG_8
  - LL_HRTIM_ADCTRIG_9
  - LL_HRTIM_ADCTRIG_10

Return values

- **Src:** This parameter can be a combination of the following values:

Reference Manual to LL API cross reference:

- ADC1R ADC1MC1 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV1 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV5 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TFC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TARST LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TFC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBRST LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TFC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TFPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TFRST LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC1 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV6 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV7 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV8 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV9 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV10 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TFC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TBC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TFC3 LL_HRTIM_GetADCTrigSrc

- ADC2R ADC2TBC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TBPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TFC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCRST LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TFPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDRST LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TEC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TEC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TEC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TERST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC1 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV1 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV5 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TFC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TARST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TFC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBRST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TFC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TFPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TFRST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MC1 LL_HRTIM_GetADCTrigSrc

- ADC4R ADC4MC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV6 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV7 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV8 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV9 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV10 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TFC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TFC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TFC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCRST LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TFPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDRST LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TEC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TEC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TEC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TERST LL_HRTIM_GetADCTrigSrc ADCER ADC5TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC6TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC7TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC8TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC9TRG LL_HRTIM_SetADCTrigSrc
- ADCER ADC10TRG LL_HRTIM_SetADCTrigSrc

LL_HRTIM_SetADCPostScaler

Function name

__STATIC_INLINE void LL_HRTIM_SetADCPostScaler (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t PostScaler)

Function description

Select the ADC post scaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
  - LL_HRTIM_ADCTRIG_5
  - LL_HRTIM_ADCTRIG_6
  - LL_HRTIM_ADCTRIG_7
  - LL_HRTIM_ADCTRIG_8
  - LL_HRTIM_ADCTRIG_9
  - LL_HRTIM_ADCTRIG_10
- **PostScaler:** This parameter can be a number between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- This function allows to adjust each ADC trigger rate individually.
- In center-aligned mode, the ADC trigger rate is also dependent on ADROM[1:0] bitfield, programmed in the source timer (see function LL_HRTIM_TIM_SetADCRollOverMode)

Reference Manual to LL API cross reference:

- ADCPS2 ADC10PSC LL_HRTIM_SetADCPostScaler
- ADCPS2 ADC9PSC LL_HRTIM_SetADCPostScaler
- ADCPS2 ADC8PSC LL_HRTIM_SetADCPostScaler
- ADCPS2 ADC7PSC LL_HRTIM_SetADCPostScaler
- ADCPS2 ADC6PSC LL_HRTIM_SetADCPostScaler
- ADCPS1 ADC5PSC LL_HRTIM_SetADCPostScaler
- ADCPS1 ADC4PSC LL_HRTIM_SetADCPostScaler
- ADCPS1 ADC3PSC LL_HRTIM_SetADCPostScaler
- ADCPS1 ADC2PSC LL_HRTIM_SetADCPostScaler
- ADCPS1 ADC1PSC LL_HRTIM_SetADCPostScaler

LL_HRTIM_GetADCPostScaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetADCPostScaler (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig)

Function description

Get the selected ADC post scaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
  - LL_HRTIM_ADCTRIG_5
  - LL_HRTIM_ADCTRIG_6
  - LL_HRTIM_ADCTRIG_7
  - LL_HRTIM_ADCTRIG_8
  - LL_HRTIM_ADCTRIG_9
  - LL_HRTIM_ADCTRIG_10

Return values

- **PostScaler:** This parameter can be a number between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- ADCPS2 ADC10PSC LL_HRTIM_GetADCPostScaler
- ADCPS2 ADC9PSC LL_HRTIM_GetADCPostScaler
- ADCPS2 ADC8PSC LL_HRTIM_GetADCPostScaler
- ADCPS2 ADC7PSC LL_HRTIM_GetADCPostScaler
- ADCPS2 ADC6PSC LL_HRTIM_GetADCPostScaler
- ADCPS1 ADC5PSC LL_HRTIM_GetADCPostScaler
- ADCPS1 ADC4PSC LL_HRTIM_GetADCPostScaler
- ADCPS1 ADC3PSC LL_HRTIM_GetADCPostScaler
- ADCPS1 ADC2PSC LL_HRTIM_GetADCPostScaler
- ADCPS1 ADC1PSC LL_HRTIM_GetADCPostScaler

LL_HRTIM_ConfigDLLCalibration

Function name

__STATIC_INLINE void LL_HRTIM_ConfigDLLCalibration (HRTIM_TypeDef * HRTIMx, uint32_t Mode, uint32_t Period)

Function description

Configure the DLL calibration mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_DLLCALIBRATION_MODE_SINGLESHOT
  - LL_HRTIM_DLLCALIBRATION_MODE_CONTINUOUS
- **Period:** This parameter can be one of the following values:
  - LL_HRTIM_DLLCALIBRATION_RATE_0
  - LL_HRTIM_DLLCALIBRATION_RATE_1
  - LL_HRTIM_DLLCALIBRATION_RATE_2
  - LL_HRTIM_DLLCALIBRATION_RATE_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- DLLCR CALEN LL_HRTIM_ConfigDLLCalibration
- DLLCR CALRTE LL_HRTIM_ConfigDLLCalibration

LL_HRTIM_StartDLLCalibration

Function name

__STATIC_INLINE void LL_HRTIM_StartDLLCalibration (HRTIM_TypeDef * HRTIMx)

Function description

Launch DLL calibration.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DLLCR CAL LL_HRTIM_StartDLLCalibration

LL_HRTIM_TIM_CounterEnable

Function name

__STATIC_INLINE void LL_HRTIM_TIM_CounterEnable (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Enable timer(s) counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER TFCEN LL_HRTIM_TIM_CounterEnable
- MDIER TECEN LL_HRTIM_TIM_CounterEnable
- MDIER TDCEN LL_HRTIM_TIM_CounterEnable
- MDIER TCCEN LL_HRTIM_TIM_CounterEnable
- MDIER TBCEN LL_HRTIM_TIM_CounterEnable
- MDIER TACEN LL_HRTIM_TIM_CounterEnable
- MDIER MCEN LL_HRTIM_TIM_CounterEnable

LL_HRTIM_TIM_CounterDisable

Function name

__STATIC_INLINE void LL_HRTIM_TIM_CounterDisable (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Disable timer(s) counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER TFCEN LL_HRTIM_TIM_CounterDisable
- MDIER TECEN LL_HRTIM_TIM_CounterDisable
- MDIER TDCEN LL_HRTIM_TIM_CounterDisable
- MDIER TCCEN LL_HRTIM_TIM_CounterDisable
- MDIER TBCEN LL_HRTIM_TIM_CounterDisable
- MDIER TACEN LL_HRTIM_TIM_CounterDisable
- MDIER MCEN LL_HRTIM_TIM_CounterDisable

LL_HRTIM_TIM_IsCounterEnabled

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsCounterEnabled (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer counter is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCEN or TxCEN bit HRTIM_MCR register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER TFCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TECEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TDCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TCCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TBCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TACEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER MCEN LL_HRTIM_TIM_IsCounterEnabled

LL_HRTIM_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Prescaler)

Function description

Set the timer clock prescaler ratio.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_PRESCALERRATIO_MUL32
  - LL_HRTIM_PRESCALERRATIO_MUL16
  - LL_HRTIM_PRESCALERRATIO_MUL8
  - LL_HRTIM_PRESCALERRATIO_MUL4
  - LL_HRTIM_PRESCALERRATIO_MUL2
  - LL_HRTIM_PRESCALERRATIO_DIV1
  - LL_HRTIM_PRESCALERRATIO_DIV2
  - LL_HRTIM_PRESCALERRATIO_DIV4

Return values

- **None:**

Notes

- The counter clock equivalent frequency (CK_CNT) is equal to fHRCK / 2^CKPSC[2:0].
- The prescaling ratio cannot be modified once the timer counter is enabled.

Reference Manual to LL API cross reference:

- MCR CKPSC LL_HRTIM_TIM_SetPrescaler
- TIMxCR CKPSC LL_HRTIM_TIM_SetPrescaler

LL_HRTIM_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the timer clock prescaler ratio.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Prescaler:** Returned value can be one of the following values:
  - LL_HRTIM_PRESCALERRATIO_MUL32
  - LL_HRTIM_PRESCALERRATIO_MUL16
  - LL_HRTIM_PRESCALERRATIO_MUL8
  - LL_HRTIM_PRESCALERRATIO_MUL4
  - LL_HRTIM_PRESCALERRATIO_MUL2
  - LL_HRTIM_PRESCALERRATIO_DIV1
  - LL_HRTIM_PRESCALERRATIO_DIV2
  - LL_HRTIM_PRESCALERRATIO_DIV4

Reference Manual to LL API cross reference:

- MCR CKPSC LL_HRTIM_TIM_GetPrescaler
- TIMxCR CKPSC LL_HRTIM_TIM_GetPrescaler

LL_HRTIM_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCounterMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Set the counter operating mode mode (single-shot, continuous or re-triggerable).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_MODE_CONTINUOUS
  - LL_HRTIM_MODE_SINGLESHOT
  - LL_HRTIM_MODE_RETRIGGERABLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR CONT LL_HRTIM_TIM_SetCounterMode
- MCR RETRIG LL_HRTIM_TIM_SetCounterMode
- TIMxCR CONT LL_HRTIM_TIM_SetCounterMode
- TIMxCR RETRIG LL_HRTIM_TIM_SetCounterMode

LL_HRTIM_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCounterMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the counter operating mode mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** Returned value can be one of the following values:
  - LL_HRTIM_MODE_CONTINUOUS
  - LL_HRTIM_MODE_SINGLESHOT
  - LL_HRTIM_MODE_RETRIGGERABLE

Reference Manual to LL API cross reference:

- MCR CONT LL_HRTIM_TIM_GetCounterMode
- MCR RETRIG LL_HRTIM_TIM_GetCounterMode
- TIMxCR CONT LL_HRTIM_TIM_GetCounterMode
- TIMxCR RETRIG LL_HRTIM_TIM_GetCounterMode

LL_HRTIM_TIM_EnableHalfMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the half duty-cycle mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- When the half mode is enabled, HRTIM_MCMP1R (or HRTIM_CMP1xR) active register is automatically updated with HRTIM_MPER/2 (or HRTIM_PERxR/2) value when HRTIM_MPER (or HRTIM_PERxR) register is written.

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_EnableHalfMode
- TIMxCR HALF LL_HRTIM_TIM_EnableHalfMode

LL_HRTIM_TIM_DisableHalfMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the half duty-cycle mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_DisableHalfMode
- TIMxCR HALF LL_HRTIM_TIM_DisableHalfMode

LL_HRTIM_TIM_IsEnabledHalfMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether half duty-cycle mode is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of HALF bit to 1 in HRTIM_MCR or HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_IsEnabledHalfMode
- TIMxCR HALF LL_HRTIM_TIM_IsEnabledHalfMode

LL_HRTIM_TIM_EnableResyncUpdate

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableResyncUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the Re-Syncronisation Update.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- The update coming from adjacent timers (when MSTU, TAU, TBU, TCU, TDU, TEU, TFU bit is set) or from a software update (TxSWU bit) is taken into account on the following reset/roll-over.

Reference Manual to LL API cross reference:

- TIMxCR RSYNCU LL_HRTIM_TIM_EnableResyncUpdate

LL_HRTIM_TIM_DisableResyncUpdate

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableResyncUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the Re-Syncronisation Update.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- The update coming from adjacent timers (when MSTU, TAU, TBU, TCU, TDU, TEU, TFU bit is set) or from a software update (TxSWU bit) is taken into account immediately.

Reference Manual to LL API cross reference:

- TIMxCR RSYNCU LL_HRTIM_TIM_DisableResyncUpdate

LL_HRTIM_TIM_IsEnabledResyncUpdate

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledResyncUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the Re-Syncronisation Update is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RSYNC bit in HRTIM_TIMxCR register (1 or 0).

Notes

- This bit specifies whether update source coming outside from the timing unit must be synchronized

Reference Manual to LL API cross reference:

- TIMxCR RSYNCU LL_HRTIM_TIM_IsEnabledResyncUpdate

LL_HRTIM_TIM_SetInterleavedMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetInterleavedMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_INTERLEAVED_MODE_DISABLED
  - LL_HRTIM_INTERLEAVED_MODE_DUAL
  - LL_HRTIM_INTERLEAVED_MODE_TRIPLE
  - LL_HRTIM_INTERLEAVED_MODE_QUAD

Return values

- **None:**

Notes

- Interleaved mode complements the Half mode and helps the implementation of interleaved topologies.
- When interleaved mode is enabled, the content of the compare registers is overridden.

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_SetInterleavedMode
- MCR INTLVD LL_HRTIM_TIM_SetInterleavedMode
- TIMxCR HALF LL_HRTIM_TIM_SetInterleavedMode
- TIMxCR INTLVD LL_HRTIM_TIM_SetInterleavedMode

LL_HRTIM_TIM_GetInterleavedMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetInterleavedMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

get the Interleaved configuration.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **This:** parameter can be one of the following values:
  - LL_HRTIM_INTERLEAVED_MODE_DISABLED
  - LL_HRTIM_INTERLEAVED_MODE_DUAL
  - LL_HRTIM_INTERLEAVED_MODE_TRIPLE
  - LL_HRTIM_INTERLEAVED_MODE_QUAD

Notes

- The interleaved Mode is Triple or Quad if HALF bit is disabled the interleaved Mode is dual if HALF bit is set,

Reference Manual to LL API cross reference:

- MCR INTLVD LL_HRTIM_TIM_GetInterleavedMode
- TIMxCR INTLVD LL_HRTIM_TIM_GetInterleavedMode

LL_HRTIM_TIM_EnableStartOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableStartOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the timer start when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSTRTM LL_HRTIM_TIM_EnableStartOnSync
- TIMxCR SYNSTRTA LL_HRTIM_TIM_EnableStartOnSync

LL_HRTIM_TIM_DisableStartOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableStartOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the timer start when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSTRTM LL_HRTIM_TIM_DisableStartOnSync
- TIMxCR SYNSTRTA LL_HRTIM_TIM_DisableStartOnSync

LL_HRTIM_TIM_IsEnabledStartOnSync

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledStartOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer start when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SYNCSTRTx bit in HRTIM_MCR or HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- MCR SYNCSTRTM LL_HRTIM_TIM_IsEnabledStartOnSync
- TIMxCR SYNSTRTA LL_HRTIM_TIM_IsEnabledStartOnSync

LL_HRTIM_TIM_EnableResetOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableResetOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the timer reset when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCRSTM LL_HRTIM_TIM_EnableResetOnSync
- TIMxCR SYNCRSTA LL_HRTIM_TIM_EnableResetOnSync

LL_HRTIM_TIM_DisableResetOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableResetOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the timer reset when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCRSTM LL_HRTIM_TIM_DisableResetOnSync
- TIMxCR SYNCRSTA LL_HRTIM_TIM_DisableResetOnSync

LL_HRTIM_TIM_IsEnabledResetOnSync

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledResetOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer reset when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCRSTM LL_HRTIM_TIM_IsEnabledResetOnSync
- TIMxCR SYNCRSTA LL_HRTIM_TIM_IsEnabledResetOnSync

LL_HRTIM_TIM_SetDACTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetDACTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t DACTrig)

Function description

Set the HRTIM output the DAC synchronization event is generated on (DACtrigOutx).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **DACTrig:** This parameter can be one of the following values:
  - LL_HRTIM_DACTRIG_NONE
  - LL_HRTIM_DACTRIG_DACTRIGOUT_1
  - LL_HRTIM_DACTRIG_DACTRIGOUT_2
  - LL_HRTIM_DACTRIG_DACTRIGOUT_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DACSYNC LL_HRTIM_TIM_SetDACTrig
- TIMxCR DACSYNC LL_HRTIM_TIM_SetDACTrig

LL_HRTIM_TIM_GetDACTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetDACTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the HRTIM output the DAC synchronization event is generated on (DACtrigOutx).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **DACTrig:** Returned value can be one of the following values:
  - LL_HRTIM_DACTRIG_NONE
  - LL_HRTIM_DACTRIG_DACTRIGOUT_1
  - LL_HRTIM_DACTRIG_DACTRIGOUT_2
  - LL_HRTIM_DACTRIG_DACTRIGOUT_3

Reference Manual to LL API cross reference:

- MCR DACSYNC LL_HRTIM_TIM_GetDACTrig
- TIMxCR DACSYNC LL_HRTIM_TIM_GetDACTrig

LL_HRTIM_TIM_EnablePreload

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnablePreload (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the timer registers preload mechanism.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- When the preload mode is enabled, accessed registers are shadow registers. Their content is transferred into the active register after an update request, either software or synchronized with an event.

Reference Manual to LL API cross reference:

- MCR PREEN LL_HRTIM_TIM_EnablePreload
- TIMxCR PREEN LL_HRTIM_TIM_EnablePreload

LL_HRTIM_TIM_DisablePreload

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisablePreload (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the timer registers preload mechanism.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR PREEN LL_HRTIM_TIM_DisablePreload
- TIMxCR PREEN LL_HRTIM_TIM_DisablePreload

LL_HRTIM_TIM_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledPreload (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer registers preload mechanism is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of PREEN bit in HRTIM_MCR or HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- MCR PREEN LL_HRTIM_TIM_IsEnabledPreload
- TIMxCR PREEN LL_HRTIM_TIM_IsEnabledPreload

LL_HRTIM_TIM_SetUpdateTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetUpdateTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t UpdateTrig)

Function description

Set the timer register update trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **UpdateTrig:** This parameter can be one of the following values:

Reference Manual to LL API cross reference:

- MCR MREPU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TAU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TBU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TCU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TDU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TEU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TFU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR MSTU LL_HRTIM_TIM_SetUpdateTrig

LL_HRTIM_TIM_GetUpdateTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetUpdateTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the timer register update trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **UpdateTrig:** Returned value can be one of the following values:

Reference Manual to LL API cross reference:

- MCR MREPU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TBU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TCU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TDU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TEU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TFU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR MSTU LL_HRTIM_TIM_GetUpdateTrig

LL_HRTIM_TIM_SetUpdateGating

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetUpdateGating (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t UpdateGating)

Function description

Set the timer registers update condition (how the registers update occurs relatively to the burst DMA transaction or an external update request received on one of the update enable inputs (UPD_EN[3:1])).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **UpdateGating:** This parameter can be one of the following values:

Reference Manual to LL API cross reference:

- MCR BRSTDMA LL_HRTIM_TIM_SetUpdateGating
- TIMxCR UPDGAT LL_HRTIM_TIM_SetUpdateGating

LL_HRTIM_TIM_GetUpdateGating

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetUpdateGating (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the timer registers update condition.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **UpdateGating:** Returned value can be one of the following values:

Reference Manual to LL API cross reference:

- MCR BRSTDMA LL_HRTIM_TIM_GetUpdateGating
- TIMxCR UPDGAT LL_HRTIM_TIM_GetUpdateGating

LL_HRTIM_TIM_EnablePushPullMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnablePushPullMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the push-pull mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxCR PSHPLL LL_HRTIM_TIM_EnablePushPullMode

LL_HRTIM_TIM_DisablePushPullMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisablePushPullMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the push-pull mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxCR PSHPLL LL_HRTIM_TIM_DisablePushPullMode

LL_HRTIM_TIM_IsEnabledPushPullMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledPushPullMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the push-pull mode is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of PSHPLL bit in HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxCR PSHPLL LL_HRTIM_TIM_IsEnabledPushPullMode
-

LL_HRTIM_TIM_SetCompareMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompareMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareUnit, uint32_t Mode)

Function description

Set the functioning mode of the compare unit (CMP2 or CMP4 can operate in standard mode or in auto delayed mode).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CompareUnit:** This parameter can be one of the following values:
  - LL_HRTIM_COMPAREUNIT_2
  - LL_HRTIM_COMPAREUNIT_4
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_COMPAREMODE_REGULAR
  - LL_HRTIM_COMPAREMODE_DELAY_NOTIMEOUT
  - LL_HRTIM_COMPAREMODE_DELAY_CMP1
  - LL_HRTIM_COMPAREMODE_DELAY_CMP3

Return values

- **None:**

Notes

- In auto-delayed mode the compare match occurs independently from the timer counter value.

Reference Manual to LL API cross reference:

- TIMxCR DELCMP2 LL_HRTIM_TIM_SetCompareMode
- TIMxCR DELCMP4 LL_HRTIM_TIM_SetCompareMode

LL_HRTIM_TIM_GetCompareMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompareMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareUnit)

Function description

Get the functioning mode of the compare unit.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CompareUnit:** This parameter can be one of the following values:
  - LL_HRTIM_COMPAREUNIT_2
  - LL_HRTIM_COMPAREUNIT_4

Return values

- **Mode:** Returned value can be one of the following values:
  - LL_HRTIM_COMPAREMODE_REGULAR
  - LL_HRTIM_COMPAREMODE_DELAY_NOTIMEOUT
  - LL_HRTIM_COMPAREMODE_DELAY_CMP1
  - LL_HRTIM_COMPAREMODE_DELAY_CMP3

Reference Manual to LL API cross reference:

- TIMxCR DELCMP2 LL_HRTIM_TIM_GetCompareMode
- TIMxCR DELCMP4 LL_HRTIM_TIM_GetCompareMode

LL_HRTIM_TIM_SetCounter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Counter)

Function description

Set the timer counter value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Counter:** Value between 0 and 0xFFFF

Return values

- **None:**

Notes

- This function can only be called when the timer is stopped.
- For HR clock prescaling ratio below 32 (CKPSC[2:0] < 5), the least significant bits of the counter are not significant. They cannot be written and return 0 when read.
- The timer behavior is not guaranteed if the counter value is set above the period.

Reference Manual to LL API cross reference:

- MCNTR MCNT LL_HRTIM_TIM_SetCounter
- CNTxR CNTx LL_HRTIM_TIM_SetCounter

LL_HRTIM_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual timer counter value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Counter:** Value between 0 and 0xFFFF

Reference Manual to LL API cross reference:

- MCNTR MCNT LL_HRTIM_TIM_GetCounter
- CNTxR CNTx LL_HRTIM_TIM_GetCounter

LL_HRTIM_TIM_SetPeriod

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetPeriod (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Period)

Function description

Set the timer period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Period:** Value between 0 and 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPER MPER LL_HRTIM_TIM_SetPeriod
- PERxR PERx LL_HRTIM_TIM_SetPeriod

LL_HRTIM_TIM_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetPeriod (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual timer period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Period:** Value between 0 and 0xFFFF

Reference Manual to LL API cross reference:

- MPER MPER LL_HRTIM_TIM_GetPeriod
- PERxR PERx LL_HRTIM_TIM_GetPeriod

LL_HRTIM_TIM_SetRepetition

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetRepetition (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Repetition)

Function description

Set the timer repetition period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Repetition:** Value between 0 and 0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MREP MREP LL_HRTIM_TIM_SetRepetition
- REPxR REPx LL_HRTIM_TIM_SetRepetition

LL_HRTIM_TIM_GetRepetition

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetRepetition (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual timer repetition period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Repetition:** Value between 0 and 0xFF

Reference Manual to LL API cross reference:

- MREP MREP LL_HRTIM_TIM_GetRepetition
- REPxR REPx LL_HRTIM_TIM_GetRepetition

LL_HRTIM_TIM_SetCompare1

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 1.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP1R MCMP1 LL_HRTIM_TIM_SetCompare1
- CMP1xR CMP1x LL_HRTIM_TIM_SetCompare1

LL_HRTIM_TIM_GetCompare1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 1.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP1R MCMP1 LL_HRTIM_TIM_GetCompare1
- CMP1xR CMP1x LL_HRTIM_TIM_GetCompare1

LL_HRTIM_TIM_SetCompare2

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 2.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP2R MCMP2 LL_HRTIM_TIM_SetCompare2
- CMP2xR CMP2x LL_HRTIM_TIM_SetCompare2

LL_HRTIM_TIM_GetCompare2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 2.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP2R MCMP2 LL_HRTIM_TIM_GetCompare2
- CMP2xR CMP2x LL_HRTIM_TIM_GetCompare2
-

LL_HRTIM_TIM_SetCompare3

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 3.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP3R MCMP3 LL_HRTIM_TIM_SetCompare3
- CMP3xR CMP3x LL_HRTIM_TIM_SetCompare3

LL_HRTIM_TIM_GetCompare3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 3.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP3R MCMP3 LL_HRTIM_TIM_GetCompare3
- CMP3xR CMP3x LL_HRTIM_TIM_GetCompare3

LL_HRTIM_TIM_SetCompare4

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 4.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP4R MCMP4 LL_HRTIM_TIM_SetCompare4
- CMP4xR CMP4x LL_HRTIM_TIM_SetCompare4

LL_HRTIM_TIM_GetCompare4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 4.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP4R MCMP4 LL_HRTIM_TIM_GetCompare4
- CMP4xR CMP4x LL_HRTIM_TIM_GetCompare4

LL_HRTIM_TIM_SetResetTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetResetTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t ResetTrig)

Function description

Set the reset trigger of a timer counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **ResetTrig:** This parameter can be a combination of the following values:
  - LL_HRTIM_RESETTRIG_NONE
  - LL_HRTIM_RESETTRIG_UPDATE
  - LL_HRTIM_RESETTRIG_CMP2
  - LL_HRTIM_RESETTRIG_CMP4
  - LL_HRTIM_RESETTRIG_MASTER_PER
  - LL_HRTIM_RESETTRIG_MASTER_CMP1
  - LL_HRTIM_RESETTRIG_MASTER_CMP2
  - LL_HRTIM_RESETTRIG_MASTER_CMP3
  - LL_HRTIM_RESETTRIG_MASTER_CMP4
  - LL_HRTIM_RESETTRIG_EEV_1
  - LL_HRTIM_RESETTRIG_EEV_2
  - LL_HRTIM_RESETTRIG_EEV_3
  - LL_HRTIM_RESETTRIG_EEV_4
  - LL_HRTIM_RESETTRIG_EEV_5
  - LL_HRTIM_RESETTRIG_EEV_6
  - LL_HRTIM_RESETTRIG_EEV_7
  - LL_HRTIM_RESETTRIG_EEV_8
  - LL_HRTIM_RESETTRIG_EEV_9
  - LL_HRTIM_RESETTRIG_EEV_10
  - LL_HRTIM_RESETTRIG_OTHER1_CMP1
  - LL_HRTIM_RESETTRIG_OTHER1_CMP2
  - LL_HRTIM_RESETTRIG_OTHER1_CMP4
  - LL_HRTIM_RESETTRIG_OTHER2_CMP1
  - LL_HRTIM_RESETTRIG_OTHER2_CMP2
  - LL_HRTIM_RESETTRIG_OTHER2_CMP4
  - LL_HRTIM_RESETTRIG_OTHER3_CMP1
  - LL_HRTIM_RESETTRIG_OTHER3_CMP2
  - LL_HRTIM_RESETTRIG_OTHER3_CMP4
  - LL_HRTIM_RESETTRIG_OTHER4_CMP1
  - LL_HRTIM_RESETTRIG_OTHER4_CMP2
  - LL_HRTIM_RESETTRIG_OTHER4_CMP4
  - LL_HRTIM_RESETTRIG_OTHER5_CMP1
  - LL_HRTIM_RESETTRIG_OTHER5_CMP2

Return values

- **None:**

Notes

- The reset of the timer counter can be triggered by up to 30 events that can be selected among the following sources: The timing unit: Compare 2, Compare 4 and Update (3 events). The master timer: Reset and Compare 1..4 (5 events). The external events EXTEVNT1..10 (10 events). All other timing units (e.g. Timer B..F for timer A): Compare 1, 2 and 4 (12 events).

Reference Manual to LL API cross reference:

- RSTxR UPDT LL_HRTIM_TIM_SetResetTrig
- RSTxR CMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR CMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTPER LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP3 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT1 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT2 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT3 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT4 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT5 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT6 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT7 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT8 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT9 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT10 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMBCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMBCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMBCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMCCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMCCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMCCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMDCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMDCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMDCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMECMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMECMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMECMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMFCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMFCMP2 LL_HRTIM_TIM_SetResetTrig

LL_HRTIM_TIM_GetResetTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetResetTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual reset trigger of a timer counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **ResetTrig:** Returned value can be one of the following values:
  - LL_HRTIM_RESETTRIG_NONE
  - LL_HRTIM_RESETTRIG_UPDATE
  - LL_HRTIM_RESETTRIG_CMP2
  - LL_HRTIM_RESETTRIG_CMP4
  - LL_HRTIM_RESETTRIG_MASTER_PER
  - LL_HRTIM_RESETTRIG_MASTER_CMP1
  - LL_HRTIM_RESETTRIG_MASTER_CMP2
  - LL_HRTIM_RESETTRIG_MASTER_CMP3
  - LL_HRTIM_RESETTRIG_MASTER_CMP4
  - LL_HRTIM_RESETTRIG_EEV_1
  - LL_HRTIM_RESETTRIG_EEV_2
  - LL_HRTIM_RESETTRIG_EEV_3
  - LL_HRTIM_RESETTRIG_EEV_4
  - LL_HRTIM_RESETTRIG_EEV_5
  - LL_HRTIM_RESETTRIG_EEV_6
  - LL_HRTIM_RESETTRIG_EEV_7
  - LL_HRTIM_RESETTRIG_EEV_8
  - LL_HRTIM_RESETTRIG_EEV_9
  - LL_HRTIM_RESETTRIG_EEV_10
  - LL_HRTIM_RESETTRIG_OTHER1_CMP1
  - LL_HRTIM_RESETTRIG_OTHER1_CMP2
  - LL_HRTIM_RESETTRIG_OTHER1_CMP4
  - LL_HRTIM_RESETTRIG_OTHER2_CMP1
  - LL_HRTIM_RESETTRIG_OTHER2_CMP2
  - LL_HRTIM_RESETTRIG_OTHER2_CMP4
  - LL_HRTIM_RESETTRIG_OTHER3_CMP1
  - LL_HRTIM_RESETTRIG_OTHER3_CMP2
  - LL_HRTIM_RESETTRIG_OTHER3_CMP4
  - LL_HRTIM_RESETTRIG_OTHER4_CMP1
  - LL_HRTIM_RESETTRIG_OTHER4_CMP2
  - LL_HRTIM_RESETTRIG_OTHER4_CMP4
  - LL_HRTIM_RESETTRIG_OTHER5_CMP1
  - LL_HRTIM_RESETTRIG_OTHER5_CMP2

Reference Manual to LL API cross reference:

- RSTxR UPDT LL_HRTIM_TIM_GetResetTrig
- RSTxR CMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR CMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTPER LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP3 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT1 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT2 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT3 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT4 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT5 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT6 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT7 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT8 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT9 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT10 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMBCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMBCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMBCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMCCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMCCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMCCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMDCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMDCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMDCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMECMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMECMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMECMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMFCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMFCMP2 LL_HRTIM_TIM_GetResetTrig

LL_HRTIM_TIM_GetCapture1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCapture1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get captured value for capture unit 1.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Captured:** value

Reference Manual to LL API cross reference:

- CPT1xR CPT1x LL_HRTIM_TIM_GetCapture1

LL_HRTIM_TIM_GetCapture1Direction

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCapture1Direction (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the counting direction when capture 1 event occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_COUNTING_MODE_UP
  - LL_HRTIM_COUNTING_MODE_UP_DOWN

Reference Manual to LL API cross reference:

- CPT1xR DIR LL_HRTIM_TIM_GetCapture1Direction

LL_HRTIM_TIM_GetCapture2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCapture2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get captured value for capture unit 2.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Captured:** value

Reference Manual to LL API cross reference:

- CPT2xR CPT2x LL_HRTIM_TIM_GetCapture2

LL_HRTIM_TIM_GetCapture2Direction

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCapture2Direction (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the counting direction when capture 2 event occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_COUNTING_MODE_UP
  - LL_HRTIM_COUNTING_MODE_UP_DOWN

Reference Manual to LL API cross reference:

- CPT2xR DIR LL_HRTIM_TIM_GetCapture2Direction

LL_HRTIM_TIM_SetCaptureTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCaptureTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CaptureUnit, uint64_t CaptureTrig)

Function description

Set the trigger of a capture unit for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CaptureUnit:** This parameter can be one of the following values:
  - LL_HRTIM_CAPTUREUNIT_1
  - LL_HRTIM_CAPTUREUNIT_2

- **CaptureTrig:** This parameter can be a combination of the following values:
  - LL_HRTIM_CAPTURETRIG_NONE
  - LL_HRTIM_CAPTURETRIG_SW
  - LL_HRTIM_CAPTURETRIG_UPDATE
  - LL_HRTIM_CAPTURETRIG_EEV_1
  - LL_HRTIM_CAPTURETRIG_EEV_2
  - LL_HRTIM_CAPTURETRIG_EEV_3
  - LL_HRTIM_CAPTURETRIG_EEV_4
  - LL_HRTIM_CAPTURETRIG_EEV_5
  - LL_HRTIM_CAPTURETRIG_EEV_6
  - LL_HRTIM_CAPTURETRIG_EEV_7
  - LL_HRTIM_CAPTURETRIG_EEV_8
  - LL_HRTIM_CAPTURETRIG_EEV_9
  - LL_HRTIM_CAPTURETRIG_EEV_10
  - LL_HRTIM_CAPTURETRIG_TA1_SET
  - LL_HRTIM_CAPTURETRIG_TA1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP2
  - LL_HRTIM_CAPTURETRIG_TB1_SET
  - LL_HRTIM_CAPTURETRIG_TB1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP2
  - LL_HRTIM_CAPTURETRIG_TC1_SET
  - LL_HRTIM_CAPTURETRIG_TC1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP2
  - LL_HRTIM_CAPTURETRIG_TD1_SET
  - LL_HRTIM_CAPTURETRIG_TD1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP2
  - LL_HRTIM_CAPTURETRIG_TE1_SET
  - LL_HRTIM_CAPTURETRIG_TE1_RESET
  - LL_HRTIM_CAPTURETRIG_TIME_CMP1
  - LL_HRTIM_CAPTURETRIG_TIME_CMP2
  - LL_HRTIM_CAPTURETRIG_TF1_SET
  - LL_HRTIM_CAPTURETRIG_TF1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMF_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMF_CMP2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPT1xCR SWCPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR UPDCPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV1CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV2CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV3CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV4CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV5CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV6CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV7CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV8CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV9CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV10CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TA1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TA1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TACMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TACMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TB1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TB1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TBCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TBCMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TC1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TC1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TCCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TCCMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TD1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TD1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TDCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TDCMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TE1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TE1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TECMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TECMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TF1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TF1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TFCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TFCMP2 LL_HRTIM_TIM_SetCaptureTrig

LL_HRTIM_TIM_GetCaptureTrig

Function name

__STATIC_INLINE uint64_t LL_HRTIM_TIM_GetCaptureTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CaptureUnit)

Function description

Get actual trigger of a capture unit for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **CaptureUnit:** This parameter can be one of the following values:
  - LL_HRTIM_CAPTUREUNIT_1
  - LL_HRTIM_CAPTUREUNIT_2

Return values

- **CaptureTrig:** This parameter can be a combination of the following values:
  - LL_HRTIM_CAPTURETRIG_NONE
  - LL_HRTIM_CAPTURETRIG_SW
  - LL_HRTIM_CAPTURETRIG_UPDATE
  - LL_HRTIM_CAPTURETRIG_EEV_1
  - LL_HRTIM_CAPTURETRIG_EEV_2
  - LL_HRTIM_CAPTURETRIG_EEV_3
  - LL_HRTIM_CAPTURETRIG_EEV_4
  - LL_HRTIM_CAPTURETRIG_EEV_5
  - LL_HRTIM_CAPTURETRIG_EEV_6
  - LL_HRTIM_CAPTURETRIG_EEV_7
  - LL_HRTIM_CAPTURETRIG_EEV_8
  - LL_HRTIM_CAPTURETRIG_EEV_9
  - LL_HRTIM_CAPTURETRIG_EEV_10
  - LL_HRTIM_CAPTURETRIG_TA1_SET
  - LL_HRTIM_CAPTURETRIG_TA1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP2
  - LL_HRTIM_CAPTURETRIG_TB1_SET
  - LL_HRTIM_CAPTURETRIG_TB1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP2
  - LL_HRTIM_CAPTURETRIG_TC1_SET
  - LL_HRTIM_CAPTURETRIG_TC1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP2
  - LL_HRTIM_CAPTURETRIG_TD1_SET
  - LL_HRTIM_CAPTURETRIG_TD1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP2
  - LL_HRTIM_CAPTURETRIG_TE1_SET
  - LL_HRTIM_CAPTURETRIG_TE1_RESET
  - LL_HRTIM_CAPTURETRIG_TIME_CMP1
  - LL_HRTIM_CAPTURETRIG_TIME_CMP2
  - LL_HRTIM_CAPTURETRIG_TF1_SET
  - LL_HRTIM_CAPTURETRIG_TF1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMF_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMF_CMP2

Reference Manual to LL API cross reference:

- CPT1xCR SWCPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR UPDCPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV1CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV2CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV3CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV4CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV5CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV6CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV7CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV8CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV9CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV10CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TA1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TA1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TACMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TACMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TB1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TB1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TBCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TBCMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TC1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TC1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TCCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TCCMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TD1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TD1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TDCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TDCMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TE1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TE1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TECMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TECMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TF1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TF1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TFCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TFCMP2 LL_HRTIM_TIM_GetCaptureTrig

LL_HRTIM_TIM_EnableDeadTime

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableDeadTime (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable deadtime insertion for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR DTEN LL_HRTIM_TIM_EnableDeadTime

LL_HRTIM_TIM_DisableDeadTime

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableDeadTime (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable deadtime insertion for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR DTEN LL_HRTIM_TIM_DisableDeadTime

LL_HRTIM_TIM_IsEnabledDeadTime

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledDeadTime (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether deadtime insertion is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DTEN bit in HRTIM_OUTxR register (1 or 0).

Reference Manual to LL API cross reference:

- OUTxR DTEN LL_HRTIM_TIM_IsEnabledDeadTime

LL_HRTIM_TIM_SetDLYPRTMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetDLYPRTMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t DLYPRTMode)

Function description

Set the delayed protection (DLYPRT) mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **DLYPRTMode:** Delayed protection (DLYPRT) mode

Notes

- This function must be called prior enabling the delayed protection
- Balanced Idle mode is only available in push-pull mode

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_SetDLYPRTMode
- OUTxR DLYPRT LL_HRTIM_TIM_SetDLYPRTMode

LL_HRTIM_TIM_GetDLYPRTMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetDLYPRTMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the delayed protection (DLYPRT) mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **DLYPRTMode:** Delayed protection (DLYPRT) mode

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_GetDLYPRTMode
- OUTxR DLYPRT LL_HRTIM_TIM_GetDLYPRTMode

LL_HRTIM_TIM_EnableDLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableDLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable delayed protection (DLYPRT) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- This function must not be called once the concerned timer is enabled

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_EnableDLYPRT

LL_HRTIM_TIM_DisableDLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableDLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable delayed protection (DLYPRT) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- This function must not be called once the concerned timer is enabled

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_DisableDLYPRT

LL_HRTIM_TIM_IsEnabledDLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledDLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether delayed protection (DLYPRT) is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DLYPRTEN bit in HRTIM_OUTxR register (1 or 0).

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_IsEnabledDLYPRT

LL_HRTIM_TIM_EnableBIAR

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableBIAR (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the Balanced Idle Automatic Resume (BIAR) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- This function must not be called once the concerned timer is enabled

Reference Manual to LL API cross reference:

- OUTxR BIAR LL_HRTIM_TIM_EnableBIAR

LL_HRTIM_TIM_DisableBIAR

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableBIAR (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the Balanced Idle Automatic Resume (BIAR) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- This function must not be called once the concerned timer is enabled

Reference Manual to LL API cross reference:

- OUTxR BIAR LL_HRTIM_TIM_DisableBIAR

LL_HRTIM_TIM_IsEnabledBIAR

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledBIAR (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the Balanced Idle Automatic Resume (BIAR) is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DLYPRTEN bit in HRTIM_OUTxR register (1 or 0).

Reference Manual to LL API cross reference:

- OUTxR BIAR LL_HRTIM_TIM_IsEnabledBIAR

LL_HRTIM_TIM_EnableFault

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Faults)

Function description

Enable the fault channel(s) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Faults:** This parameter can be a combination of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTxR FLT1EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT2EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT3EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT4EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT5EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT6EN LL_HRTIM_TIM_EnableFault

LL_HRTIM_TIM_DisableFault

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Faults)

Function description

Disable the fault channel(s) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Faults:** This parameter can be a combination of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTxR FLT1EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT2EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT3EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT4EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT5EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT6EN LL_HRTIM_TIM_DisableFault

LL_HRTIM_TIM_IsEnabledFault

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Fault)

Function description

Indicate whether the fault channel is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **State:** of FLTxEN bit in HRTIM_FLTxR register (1 or 0).

Reference Manual to LL API cross reference:

- FLTxR FLT1EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT2EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT3EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT4EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT5EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT6EN LL_HRTIM_TIM_IsEnabledFault

LL_HRTIM_TIM_LockFault

Function name

__STATIC_INLINE void LL_HRTIM_TIM_LockFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the fault conditioning set-up for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- Timer fault-related set-up is frozen until the next HRTIM or system reset

Reference Manual to LL API cross reference:

- FLTxR FLTLCK LL_HRTIM_TIM_LockFault

LL_HRTIM_TIM_SetBurstModeOption

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetBurstModeOption (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t BurtsModeOption)

Function description

Define how the timer behaves during a burst mode operation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **BurtsModeOption:** This parameter can be one of the following values:
  - LL_HRTIM_BURSTMODE_MAINTAINCLOCK
  - LL_HRTIM_BURSTMODE_RESETCOUNTER

Return values

- **None:**

Notes

- This function must not be called when the burst mode is enabled

Reference Manual to LL API cross reference:

- BMCR MTBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TABM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TBBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TCBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TDBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TEBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TFBM LL_HRTIM_TIM_SetBurstModeOption

LL_HRTIM_TIM_GetBurstModeOption

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetBurstModeOption (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Retrieve how the timer behaves during a burst mode operation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **BurtsMode:** This parameter can be one of the following values:
  - LL_HRTIM_BURSTMODE_MAINTAINCLOCK
  - LL_HRTIM_BURSTMODE_RESETCOUNTER

Reference Manual to LL API cross reference:

- BMCR MCR LL_HRTIM_TIM_GetBurstModeOption
- BMCR TABM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TBBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TCBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TDBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TEBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TFBM LL_HRTIM_TIM_GetBurstModeOption

LL_HRTIM_TIM_ConfigBurstDMA

Function name

__STATIC_INLINE void LL_HRTIM_TIM_ConfigBurstDMA (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Registers)

Function description

Program which registers are to be written by Burst DMA transfers.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Registers:** Registers to be updated by the DMA request

Reference Manual to LL API cross reference:

- BDMUPDR MTBM LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MICR LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MDIER LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCNT LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MPER LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MREP LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP1 LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP2 LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP3 LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP4 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxICR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxDIER LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCNT LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxPER LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxREP LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP1 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP2 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP3 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP4 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxDTR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxSET1R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxRST1R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxSET2R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxRST2R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxEEFR1 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxEEFR2 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxRSTR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxOUTR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxLTCH LL_HRTIM_TIM_ConfigBurstDMA

LL_HRTIM_TIM_GetCurrentPushPullStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCurrentPushPullStatus (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate on which output the signal is currently applied.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **CPPSTAT:** This parameter can be one of the following values:
  - LL_HRTIM_CPPSTAT_OUTPUT1
  - LL_HRTIM_CPPSTAT_OUTPUT2

Notes

- Only significant when the timer operates in push-pull mode.

Reference Manual to LL API cross reference:

- TIMxISR CPPSTAT LL_HRTIM_TIM_GetCurrentPushPullStatus

LL_HRTIM_TIM_GetIdlePushPullStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetIdlePushPullStatus (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate on which output the signal was applied, in push-pull mode, balanced fault mode or delayed idle mode, when the protection was triggered.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **IPPSTAT:** This parameter can be one of the following values:
  - LL_HRTIM_IPPSTAT_OUTPUT1
  - LL_HRTIM_IPPSTAT_OUTPUT2

Reference Manual to LL API cross reference:

- TIMxISR IPPSTAT LL_HRTIM_TIM_GetIdlePushPullStatus

LL_HRTIM_TIM_SetEventFilter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventFilter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event, uint32_t Filter)

Function description

Set the event filter for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EEFLTR_NONE
  - LL_HRTIM_EEFLTR_BLANKINGCMP1
  - LL_HRTIM_EEFLTR_BLANKINGCMP2
  - LL_HRTIM_EEFLTR_BLANKINGCMP3
  - LL_HRTIM_EEFLTR_BLANKINGCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF1_TIMBCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF2_TIMBCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF3_TIMBOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF4_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF5_TIMCCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF6_TIMFCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF7_TIMDCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF8_TIMECMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF1_TIMACMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF2_TIMACMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF3_TIMAOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF4_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF5_TIMCCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF6_TIMFCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF7_TIMDCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF8_TIMECMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF1_TIMACMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF2_TIMBCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF3_TIMBCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF4_TIMFCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF5_TIMDCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF6_TIMDCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF7_TIMDOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF8_TIMECMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF1_TIMACMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF2_TIMBCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF3_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF4_TIMCCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF5_TIMCOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF6_TIMECMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF7_TIMECMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF8_TIMFCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF1_TIMACMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF2_TIMBCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF3_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF4_TIMFCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF5_TIMFOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF6_TIMDCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF7_TIMDCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF8_TIMDOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF1_TIMACMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF2_TIMBCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF3_TIMCCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF4_TIMDCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF5_TIMDCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF6_TIMECMP1

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.

Reference Manual to LL API cross reference:

- EEFxR1 EE1LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE2LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE3LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE4LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE5LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE6LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE7LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE8LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE9LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE10LTCH LL_HRTIM_TIM_SetEventFilter

LL_HRTIM_TIM_GetEventFilter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventFilter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event)

Function description

Get actual event filter settings for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EEFLTR_NONE
  - LL_HRTIM_EEFLTR_BLANKINGCMP1
  - LL_HRTIM_EEFLTR_BLANKINGCMP2
  - LL_HRTIM_EEFLTR_BLANKINGCMP3
  - LL_HRTIM_EEFLTR_BLANKINGCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF1_TIMBCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF2_TIMBCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF3_TIMBOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF4_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF5_TIMCCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF6_TIMFCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF7_TIMDCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMAEEF8_TIMECMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF1_TIMACMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF2_TIMACMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF3_TIMAOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF4_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF5_TIMCCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF6_TIMFCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF7_TIMDCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMBEEF8_TIMECMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF1_TIMACMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF2_TIMBCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF3_TIMBCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF4_TIMFCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF5_TIMDCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF6_TIMDCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF7_TIMDOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMCEEF8_TIMECMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF1_TIMACMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF2_TIMBCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF3_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF4_TIMCCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF5_TIMCOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF6_TIMECMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF7_TIMECMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMDEEF8_TIMFCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF1_TIMACMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF2_TIMBCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF3_TIMCCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF4_TIMFCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF5_TIMFOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF6_TIMDCMP1
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF7_TIMDCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMEEEF8_TIMDOUT2
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF1_TIMACMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF2_TIMBCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF3_TIMCCMP4
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF4_TIMDCMP2
  - LL_HRTIM_EEFLTR_BLANKING_TIMFEEF5_TIMDCMP4

Reference Manual to LL API cross reference:

- EEFxR1 EE1FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE2FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE3FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE4FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE5FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE6FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE7FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE8FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE9FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE10FLTR LL_HRTIM_TIM_GetEventFilter

LL_HRTIM_TIM_SetEventLatchStatus

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventLatchStatus (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event, uint32_t LatchStatus)

Function description

Enable or disable event latch mechanism for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **LatchStatus:** This parameter can be one of the following values:
  - LL_HRTIM_EELATCH_DISABLED
  - LL_HRTIM_EELATCH_ENABLED

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.

Reference Manual to LL API cross reference:

- EEFxR1 EE1LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE2LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE3LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE4LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE5LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE6LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE7LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE8LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE9LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE10LTCH LL_HRTIM_TIM_SetEventLatchStatus

LL_HRTIM_TIM_GetEventLatchStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventLatchStatus (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event)

Function description

Get actual event latch status for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **LatchStatus:** This parameter can be one of the following values:
  - LL_HRTIM_EELATCH_DISABLED
  - LL_HRTIM_EELATCH_ENABLED

Reference Manual to LL API cross reference:

- EEFxR1 EE1LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE2LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE3LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE4LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE5LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE6LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE7LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE8LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE9LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE10LTCH LL_HRTIM_TIM_GetEventLatchStatus

LL_HRTIM_TIM_SetTriggeredHalfMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetTriggeredHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the Trigger-Half operating mode for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_TRIGHALF_ENABLED
  - LL_HRTIM_TRIGHALF_DISABLED

Return values

- **None:**

Notes

- This bitfield defines whether the compare 2 register
- is behaving in standard mode (compare match issued as soon as counter equal compare)
- or in triggered-half mode

Reference Manual to LL API cross reference:

- TIMxCR2 TRGHLF LL_HRTIM_TIM_SetTriggeredHalfMode

LL_HRTIM_TIM_GetTriggeredHalfMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetTriggeredHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the Trigger-Half operating mode for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_TRIGHALF_ENABLED
  - LL_HRTIM_TRIGHALF_DISABLED

Notes

- This bitfield reports whether the compare 2 register
- is behaving in standard mode (compare match issued as soon as counter equal compare)
- or in triggered-half mode

Reference Manual to LL API cross reference:

- TIMxCR2 TRGHLF LL_HRTIM_TIM_GetTriggeredHalfMode

LL_HRTIM_TIM_SetComp1Mode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetComp1Mode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the compare 1 operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_GTCMP1_EQUAL
  - LL_HRTIM_GTCMP1_GREATER

Return values

- **None:**

Notes

- This bit defines the compare 1 operating mode:
- 0: the compare 1 event is generated when the counter is equal to the compare value
- 1: the compare 1 event is generated when the counter is greater than the compare value

Reference Manual to LL API cross reference:

- TIMxCR2 GTCMP1 LL_HRTIM_TIM_SetComp1Mode

LL_HRTIM_TIM_GetComp1Mode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetComp1Mode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the selected compare 1 operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_GTCMP1_EQUAL
  - LL_HRTIM_GTCMP1_GREATER

Notes

- This bit reports the compare 1 operating mode:
- 0: the compare 1 event is generated when the counter is equal to the compare value
- 1: the compare 1 event is generated when the counter is greater than the compare value

Reference Manual to LL API cross reference:

- TIMxCR2 GTCMP1 LL_HRTIM_TIM_GetComp1Mode

LL_HRTIM_TIM_SetComp3Mode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetComp3Mode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the compare 3 operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_GTCMP3_EQUAL
  - LL_HRTIM_GTCMP3_GREATER

Return values

- **None:**

Notes

- This bit defines the compare 3 operating mode:
- 0: the compare 3 event is generated when the counter is equal to the compare value
- 1: the compare 3 event is generated when the counter is greater than the compare value

Reference Manual to LL API cross reference:

- TIMxCR2 GTCMP3 LL_HRTIM_TIM_SetComp3Mode

LL_HRTIM_TIM_GetComp3Mode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetComp3Mode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the selected compare 3 operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_GTCMP3_EQUAL
  - LL_HRTIM_GTCMP3_GREATER

Notes

- This bit reports the compare 3 operating mode:
- 0: the compare 3 event is generated when the counter is equal to the compare value
- 1: the compare 3 event is generated when the counter is greater than the compare value

Reference Manual to LL API cross reference:

- TIMxCR2 GTCMP3 LL_HRTIM_TIM_GetComp1Mode

LL_HRTIM_TIM_SetRollOverMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Return values

- **None:**

Notes

- Only significant in up-down counting mode (see function LL_HRTIM_TIM_SetCountingMode()).
- Only concerns the Roll-over event with the following destinations: Update trigger, IRQ and DMA requests, repetition counter decrement and External Event filtering.

Reference Manual to LL API cross reference:

- TIMxCR2 ROM LL_HRTIM_TIM_SetRollOverMode

LL_HRTIM_TIM_GetRollOverMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected the roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** returned value can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Reference Manual to LL API cross reference:

- TIMxCR2 ROM LL_HRTIM_TIM_GetRollOverMode

LL_HRTIM_TIM_SetFaultEventRollOverMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetFaultEventRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select Fault and Event roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Return values

- **None:**

Notes

- Only significant in up-down counting mode (see function LL_HRTIM_TIM_SetCountingMode()).
- only concerns the Roll-over event used by the Fault and Event counters.

Reference Manual to LL API cross reference:

- TIMxCR2 FEROM LL_HRTIM_TIM_SetFaultEventRollOverMode

LL_HRTIM_TIM_GetFaultEventRollOverMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetFaultEventRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected Fault and Event role-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** returned value can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Reference Manual to LL API cross reference:

- TIMxCR2 FEROM LL_HRTIM_TIM_GetFaultEventRollOverMode

LL_HRTIM_TIM_SetBMRollOverMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetBMRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the Burst mode roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Return values

- **None:**

Notes

- Only significant in up-down counting mode (see function LL_HRTIM_TIM_SetCountingMode()).
- Only concerns the Roll-over event used in the Burst mode controller, as clock as as burst mode trigger.

Reference Manual to LL API cross reference:

- TIMxCR2 BMROM LL_HRTIM_TIM_SetBMRollOverMode

LL_HRTIM_TIM_GetBMRollOverMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetBMRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected Burst mode roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** returned value can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Reference Manual to LL API cross reference:

- TIMxCR2 ROM LL_HRTIM_TIM_GetBMRollOverMode

LL_HRTIM_TIM_SetADCRollOverMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetADCRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the ADC roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Return values

- **None:**

Notes

- Only significant in up-down counting mode (see function LL_HRTIM_TIM_SetCountingMode()).
- Only concerns the Roll-over event used to trigger the ADC.

Reference Manual to LL API cross reference:

- TIMxCR2 BMROM LL_HRTIM_TIM_SetADCRollOverMode

LL_HRTIM_TIM_GetADCRollOverMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetADCRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected ADC roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** returned value can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Reference Manual to LL API cross reference:

- TIMxCR2 BMROM LL_HRTIM_TIM_GetADCRollOverMode

LL_HRTIM_TIM_SetOutputRollOverMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetOutputRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the ADC roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Return values

- **None:**

Notes

- Only significant in up-down counting mode (see function LL_HRTIM_TIM_SetCountingMode()).
- Only concerns concerns the Roll-over event which sets and/or resets the ouputs, as per HRTIM_SETxyR and HRTIM_RSTxyR settings (see function LL_HRTIM_OUT_SetOutputSetSrc() and function LL_HRTIM_OUT_SetOutputResetSrc() respectively).

Reference Manual to LL API cross reference:

- TIMxCR2 OUTROM LL_HRTIM_TIM_SetOutputRollOverMode

LL_HRTIM_TIM_GetOutputRollOverMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetOutputRollOverMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected ADC roll-over mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** returned value can be one of the following values:
  - LL_HRTIM_ROLLOVER_MODE_PER
  - LL_HRTIM_ROLLOVER_MODE_RST
  - LL_HRTIM_ROLLOVER_MODE_BOTH

Reference Manual to LL API cross reference:

- TIMxCR2 OUTROM LL_HRTIM_TIM_GetOutputRollOverMode

LL_HRTIM_TIM_SetCountingMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCountingMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select the counting mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_COUNTING_MODE_UP
  - LL_HRTIM_COUNTING_MODE_UP_DOWN

Return values

- **None:**

Notes

- The up-down counting mode is available for both continuous and single-shot (retriggerable and nonretriggerable) operating modes (see function LL_HRTIM_TIM_SetCounterMode()).
- The counter roll-over event is defined differently in-up-down counting mode to support various operating condition. See LL_HRTIM_TIM_SetCounterMode()

Reference Manual to LL API cross reference:

- TIMxCR2 UDM LL_HRTIM_TIM_SetCountingMode

LL_HRTIM_TIM_GetCountingMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCountingMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected counting mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Mode:** returned value can be one of the following values:
  - LL_HRTIM_COUNTING_MODE_UP
  - LL_HRTIM_COUNTING_MODE_UP_DOWN
- **None:**

Reference Manual to LL API cross reference:

- TIMxCR2 UDM LL_HRTIM_TIM_GetCountingMode

LL_HRTIM_TIM_SetDualDacResetTrigger

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetDualDacResetTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select Dual Channel DAC Reset trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_DCDR_COUNTER
  - LL_HRTIM_DCDR_OUT1SET

Return values

- **None:**

Notes

- Significant only when Dual channel DAC trigger is enabled (see function LL_HRTIM_TIM_EnableDualDacTrigger()).

Reference Manual to LL API cross reference:

- TIMxCR2 DCDR LL_HRTIM_TIM_SetDualDacResetTrigger

LL_HRTIM_TIM_GetDualDacResetTrigger

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetDualDacResetTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected Dual Channel DAC Reset trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Trigger:** returned value can be one of the following values:
  - LL_HRTIM_DCDR_COUNTER
  - LL_HRTIM_DCDR_OUT1SET

Reference Manual to LL API cross reference:

- TIMxCR2 DCDR LL_HRTIM_TIM_GetDualDacResetTrigger

LL_HRTIM_TIM_SetDualDacStepTrigger

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetDualDacStepTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Select Dual Channel DAC Reset trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_DCDS_CMP2
  - LL_HRTIM_DCDS_OUT1RST

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxCR2 DCDS LL_HRTIM_TIM_SetDualDacStepTrigger

LL_HRTIM_TIM_GetDualDacStepTrigger

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetDualDacStepTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get selected Dual Channel DAC Reset trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Trigger:** returned value can be one of the following values:
  - LL_HRTIM_DCDS_CMP2
  - LL_HRTIM_DCDS_OUT1RST

Reference Manual to LL API cross reference:

- TIMxCR2 DCDS LL_HRTIM_TIM_GetDualDacStepTrigger

LL_HRTIM_TIM_EnableDualDacTrigger

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableDualDacTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable Dual Channel DAC trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Notes

- Only significant when balanced Idle mode is enabled (see function LL_HRTIM_TIM_SetDLYPRTMode()).

Reference Manual to LL API cross reference:

- TIMxCR2 DCDE LL_HRTIM_TIM_EnableDualDacTrigger

LL_HRTIM_TIM_DisableDualDacTrigger

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableDualDacTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable Dual Channel DAC trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxCR2 DCDE LL_HRTIM_TIM_DisableDualDacTrigger

LL_HRTIM_TIM_IsEnabledDualDacTrigger

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledDualDacTrigger (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether Dual Channel DAC trigger is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DCDE bit in HRTIM_TIMxCR2 register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxCR2 DCDE LL_HRTIM_TIM_IsEnabledDualDacTrigger

LL_HRTIM_TIM_SetEventCounterThreshold

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventCounterThreshold (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter, uint32_t Threshold)

Function description

Set the external event counter threshold.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B
- **Threshold:** This parameter can be a number between Min_Data=0 and Max_Data=63

Return values

- **None:**

Notes

- The external event is propagated to the timer only if the number of active edges is greater than the external event counter threshold.

Reference Manual to LL API cross reference:

- EEFxR3 EEVBCNT LL_HRTIM_TIM_SetEventCounterThreshold
- EEFxR3 EEVACNT LL_HRTIM_TIM_SetEventCounterThreshold

LL_HRTIM_TIM_GetEventCounterThreshold

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventCounterThreshold (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

Get the programmed external event counter threshold.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **Threshold:** Value between Min_Data=0 and Max_Data=63

Reference Manual to LL API cross reference:

- EEFxR3 EEVBCNT LL_HRTIM_TIM_GetEventCounterThreshold
- EEFxR3 EEVACNT LL_HRTIM_TIM_GetEventCounterThreshold

LL_HRTIM_TIM_SetEventCounterSource

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventCounterSource (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter, uint32_t Event)

Function description

Select the external event counter source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **None:**

Notes

- External event counting is only valid for edge-sensitive external events (See function LL_HRTIM_EE_Config() and function LL_HRTIM_EE_SetSensitivity()).

Reference Manual to LL API cross reference:

- EEFxR3 EEVBSEL LL_HRTIM_TIM_SetEventCounterSource
- EEFxR3 EEVASEL LL_HRTIM_TIM_SetEventCounterSource

LL_HRTIM_TIM_GetEventCounterSource

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventCounterSource (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

get the selected external event counter source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Reference Manual to LL API cross reference:

- EEFxR3 EEVBSEL LL_HRTIM_TIM_GetEventCounterSource
- EEFxR3 EEVASEL LL_HRTIM_TIM_GetEventCounterSource

LL_HRTIM_TIM_SetEventCounterResetMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventCounterResetMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter, uint32_t Mode)

Function description

Select the external event counter reset mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTERRSTMODE_UNCONDITIONAL
  - LL_HRTIM_EVENT_COUNTERRSTMODE_CONDITIONAL

Return values

- **None:**

Reference Manual to LL API cross reference:

- EEFxR3 EEVBRSTM LL_HRTIM_TIM_SetEventCounterResetMode
- EEFxR3 EEVARSTM LL_HRTIM_TIM_SetEventCounterResetMode

LL_HRTIM_TIM_GetEventCounterResetMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventCounterResetMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

Get selected external event counter reset mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTERRSTMODE_UNCONDITIONAL
  - LL_HRTIM_EVENT_COUNTERRSTMODE_CONDITIONAL

Reference Manual to LL API cross reference:

- EEFxR3 EEVBRSTM LL_HRTIM_TIM_GetEventCounterResetMode
- EEFxR3 EEVARSTM LL_HRTIM_TIM_GetEventCounterResetMode

LL_HRTIM_TIM_ResetEventCounter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_ResetEventCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

Reset the external event counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **None:**

Reference Manual to LL API cross reference:

- EEFxR3 EEVACRES LL_HRTIM_TIM_ResetEventCounter
- EEFxR3 EEVBCRES LL_HRTIM_TIM_ResetEventCounter

LL_HRTIM_TIM_EnableEventCounter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableEventCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

Enable the external event counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **None:**

Reference Manual to LL API cross reference:

- EEFxR3 EEVACE LL_HRTIM_TIM_EnableEventCounter
- EEFxR3 EEVBCE LL_HRTIM_TIM_EnableEventCounter

LL_HRTIM_TIM_DisableEventCounter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableEventCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

Disable the external event counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **None:**

Reference Manual to LL API cross reference:

- EEFxR3 EEVACE LL_HRTIM_TIM_DisableEventCounter
- EEFxR3 EEVBCE LL_HRTIM_TIM_DisableEventCounter

LL_HRTIM_TIM_IsEnabledEventCounter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledEventCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t EventCounter)

Function description

Indicate whether the external event counter is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **EventCounter:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_COUNTER_A
  - LL_HRTIM_EVENT_COUNTER_B

Return values

- **State:** of EEVxCE bit in RTIM_EEFxR3 register (1 or 0).

Reference Manual to LL API cross reference:

- EEFxR3 EEVACE LL_HRTIM_TIM_IsEnabledEventCounter
- EEFxR3 EEVBCE LL_HRTIM_TIM_IsEnabledEventCounter

LL_HRTIM_DT_Config

Function name

__STATIC_INLINE void LL_HRTIM_DT_Config (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Configuration)

Function description

Configure the dead time insertion feature for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_DT_PRESCALER_MUL8 or ... or LL_HRTIM_DT_PRESCALER_DIV16
  - LL_HRTIM_DT_RISING_POSITIVE or LL_HRTIM_DT_RISING_NEGATIVE
  - LL_HRTIM_DT_FALLING_POSITIVE or LL_HRTIM_DT_FALLING_NEGATIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTPRSC LL_HRTIM_DT_Config
- DTxR SDTF LL_HRTIM_DT_Config
- DTxR SDRT LL_HRTIM_DT_Config

LL_HRTIM_DT_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Prescaler)

Function description

Set the deadtime prescaler value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_DT_PRESCALER_MUL8
  - LL_HRTIM_DT_PRESCALER_MUL4
  - LL_HRTIM_DT_PRESCALER_MUL2
  - LL_HRTIM_DT_PRESCALER_DIV1
  - LL_HRTIM_DT_PRESCALER_DIV2
  - LL_HRTIM_DT_PRESCALER_DIV4
  - LL_HRTIM_DT_PRESCALER_DIV8
  - LL_HRTIM_DT_PRESCALER_DIV16

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTPRSC LL_HRTIM_DT_SetPrescaler

LL_HRTIM_DT_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime prescaler value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_DT_PRESCALER_MUL8
  - LL_HRTIM_DT_PRESCALER_MUL4
  - LL_HRTIM_DT_PRESCALER_MUL2
  - LL_HRTIM_DT_PRESCALER_DIV1
  - LL_HRTIM_DT_PRESCALER_DIV2
  - LL_HRTIM_DT_PRESCALER_DIV4
  - LL_HRTIM_DT_PRESCALER_DIV8
  - LL_HRTIM_DT_PRESCALER_DIV16

Reference Manual to LL API cross reference:

- DTxR DTPRSC LL_HRTIM_DT_GetPrescaler

LL_HRTIM_DT_SetRisingValue

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetRisingValue (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t RisingValue)

Function description

Set the deadtime rising value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **RisingValue:** Value between 0 and 0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTR LL_HRTIM_DT_SetRisingValue

LL_HRTIM_DT_GetRisingValue

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetRisingValue (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime rising value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **RisingValue:** Value between 0 and 0x1FF

Reference Manual to LL API cross reference:

- DTxR DTR LL_HRTIM_DT_GetRisingValue

LL_HRTIM_DT_SetRisingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetRisingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t RisingSign)

Function description

Set the deadtime sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **RisingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_RISING_POSITIVE
  - LL_HRTIM_DT_RISING_NEGATIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR SDTR LL_HRTIM_DT_SetRisingSign

LL_HRTIM_DT_GetRisingSign

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetRisingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **RisingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_RISING_POSITIVE
  - LL_HRTIM_DT_RISING_NEGATIVE

Reference Manual to LL API cross reference:

- DTxR SDTR LL_HRTIM_DT_GetRisingSign

LL_HRTIM_DT_SetFallingValue

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetFallingValue (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t FallingValue)

Function description

Set the deadime falling value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **FallingValue:** Value between 0 and 0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTF LL_HRTIM_DT_SetFallingValue

LL_HRTIM_DT_GetFallingValue

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetFallingValue (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime falling value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **FallingValue:** Value between 0 and 0x1FF

Reference Manual to LL API cross reference:

- DTxR DTF LL_HRTIM_DT_GetFallingValue

LL_HRTIM_DT_SetFallingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetFallingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t FallingSign)

Function description

Set the deadtime sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **FallingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_FALLING_POSITIVE
  - LL_HRTIM_DT_FALLING_NEGATIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR SDTF LL_HRTIM_DT_SetFallingSign

LL_HRTIM_DT_GetFallingSign

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetFallingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **FallingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_FALLING_POSITIVE
  - LL_HRTIM_DT_FALLING_NEGATIVE

Reference Manual to LL API cross reference:

- DTxR SDTF LL_HRTIM_DT_GetFallingSign

LL_HRTIM_DT_LockRising

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockRising (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime value and sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTRLK LL_HRTIM_DT_LockRising

LL_HRTIM_DT_LockRisingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockRisingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTRSLK LL_HRTIM_DT_LockRisingSign

LL_HRTIM_DT_LockFalling

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockFalling (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime value and sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTFLK LL_HRTIM_DT_LockFalling

LL_HRTIM_DT_LockFallingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockFallingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTFSLK LL_HRTIM_DT_LockFallingSign

LL_HRTIM_CHP_Config

Function name

__STATIC_INLINE void LL_HRTIM_CHP_Config (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Configuration)

Function description

Configure the chopper stage for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_CHP_PRESCALER_DIV16 or ... or LL_HRTIM_CHP_PRESCALER_DIV256
  - LL_HRTIM_CHP_DUTYCYCLE_0 or ... or LL_HRTIM_CHP_DUTYCYCLE_875
  - LL_HRTIM_CHP_PULSEWIDTH_16 or ... or LL_HRTIM_CHP_PULSEWIDTH_256

Return values

- **None:**

Notes

- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR CARFRQ LL_HRTIM_CHP_Config
- CHPxR CARDTY LL_HRTIM_CHP_Config
- CHPxR STRTPW LL_HRTIM_CHP_Config

LL_HRTIM_CHP_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_CHP_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Prescaler)

Function description

Set prescaler determining the carrier frequency to be added on top of the timer output signals when chopper mode is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PRESCALER_DIV16
  - LL_HRTIM_CHP_PRESCALER_DIV32
  - LL_HRTIM_CHP_PRESCALER_DIV48
  - LL_HRTIM_CHP_PRESCALER_DIV64
  - LL_HRTIM_CHP_PRESCALER_DIV80
  - LL_HRTIM_CHP_PRESCALER_DIV96
  - LL_HRTIM_CHP_PRESCALER_DIV112
  - LL_HRTIM_CHP_PRESCALER_DIV128
  - LL_HRTIM_CHP_PRESCALER_DIV144
  - LL_HRTIM_CHP_PRESCALER_DIV160
  - LL_HRTIM_CHP_PRESCALER_DIV176
  - LL_HRTIM_CHP_PRESCALER_DIV192
  - LL_HRTIM_CHP_PRESCALER_DIV208
  - LL_HRTIM_CHP_PRESCALER_DIV224
  - LL_HRTIM_CHP_PRESCALER_DIV240
  - LL_HRTIM_CHP_PRESCALER_DIV256

Return values

- **None:**

Notes

- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR CARFRQ LL_HRTIM_CHP_SetPrescaler

LL_HRTIM_CHP_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_CHP_GetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual chopper stage prescaler value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PRESCALER_DIV16
  - LL_HRTIM_CHP_PRESCALER_DIV32
  - LL_HRTIM_CHP_PRESCALER_DIV48
  - LL_HRTIM_CHP_PRESCALER_DIV64
  - LL_HRTIM_CHP_PRESCALER_DIV80
  - LL_HRTIM_CHP_PRESCALER_DIV96
  - LL_HRTIM_CHP_PRESCALER_DIV112
  - LL_HRTIM_CHP_PRESCALER_DIV128
  - LL_HRTIM_CHP_PRESCALER_DIV144
  - LL_HRTIM_CHP_PRESCALER_DIV160
  - LL_HRTIM_CHP_PRESCALER_DIV176
  - LL_HRTIM_CHP_PRESCALER_DIV192
  - LL_HRTIM_CHP_PRESCALER_DIV208
  - LL_HRTIM_CHP_PRESCALER_DIV224
  - LL_HRTIM_CHP_PRESCALER_DIV240
  - LL_HRTIM_CHP_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- CHPxR CARFRQ LL_HRTIM_CHP_GetPrescaler

LL_HRTIM_CHP_SetDutyCycle

Function name

__STATIC_INLINE void LL_HRTIM_CHP_SetDutyCycle (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t DutyCycle)

Function description

Set the chopper duty cycle.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **DutyCycle:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_DUTYCYCLE_0
  - LL_HRTIM_CHP_DUTYCYCLE_125
  - LL_HRTIM_CHP_DUTYCYCLE_250
  - LL_HRTIM_CHP_DUTYCYCLE_375
  - LL_HRTIM_CHP_DUTYCYCLE_500
  - LL_HRTIM_CHP_DUTYCYCLE_625
  - LL_HRTIM_CHP_DUTYCYCLE_750
  - LL_HRTIM_CHP_DUTYCYCLE_875

Return values

- **None:**

Notes

- Duty cycle can be adjusted by 1/8 step (from 0/8 up to 7/8)
- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR CARDTY LL_HRTIM_CHP_SetDutyCycle

LL_HRTIM_CHP_GetDutyCycle

Function name

__STATIC_INLINE uint32_t LL_HRTIM_CHP_GetDutyCycle (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual chopper duty cycle.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **DutyCycle:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_DUTYCYCLE_0
  - LL_HRTIM_CHP_DUTYCYCLE_125
  - LL_HRTIM_CHP_DUTYCYCLE_250
  - LL_HRTIM_CHP_DUTYCYCLE_375
  - LL_HRTIM_CHP_DUTYCYCLE_500
  - LL_HRTIM_CHP_DUTYCYCLE_625
  - LL_HRTIM_CHP_DUTYCYCLE_750
  - LL_HRTIM_CHP_DUTYCYCLE_875

Reference Manual to LL API cross reference:

- CHPxR CARDTY LL_HRTIM_CHP_GetDutyCycle

LL_HRTIM_CHP_SetPulseWidth

Function name

__STATIC_INLINE void LL_HRTIM_CHP_SetPulseWidth (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t PulseWidth)

Function description

Set the start pulse width.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F
- **PulseWidth:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PULSEWIDTH_16
  - LL_HRTIM_CHP_PULSEWIDTH_32
  - LL_HRTIM_CHP_PULSEWIDTH_48
  - LL_HRTIM_CHP_PULSEWIDTH_64
  - LL_HRTIM_CHP_PULSEWIDTH_80
  - LL_HRTIM_CHP_PULSEWIDTH_96
  - LL_HRTIM_CHP_PULSEWIDTH_112
  - LL_HRTIM_CHP_PULSEWIDTH_128
  - LL_HRTIM_CHP_PULSEWIDTH_144
  - LL_HRTIM_CHP_PULSEWIDTH_160
  - LL_HRTIM_CHP_PULSEWIDTH_176
  - LL_HRTIM_CHP_PULSEWIDTH_192
  - LL_HRTIM_CHP_PULSEWIDTH_208
  - LL_HRTIM_CHP_PULSEWIDTH_224
  - LL_HRTIM_CHP_PULSEWIDTH_240
  - LL_HRTIM_CHP_PULSEWIDTH_256

Return values

- **None:**

Notes

- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR STRPW LL_HRTIM_CHP_SetPulseWidth

LL_HRTIM_CHP_GetPulseWidth

Function name

__STATIC_INLINE uint32_t LL_HRTIM_CHP_GetPulseWidth (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual start pulse width.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **PulseWidth:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PULSEWIDTH_16
  - LL_HRTIM_CHP_PULSEWIDTH_32
  - LL_HRTIM_CHP_PULSEWIDTH_48
  - LL_HRTIM_CHP_PULSEWIDTH_64
  - LL_HRTIM_CHP_PULSEWIDTH_80
  - LL_HRTIM_CHP_PULSEWIDTH_96
  - LL_HRTIM_CHP_PULSEWIDTH_112
  - LL_HRTIM_CHP_PULSEWIDTH_128
  - LL_HRTIM_CHP_PULSEWIDTH_144
  - LL_HRTIM_CHP_PULSEWIDTH_160
  - LL_HRTIM_CHP_PULSEWIDTH_176
  - LL_HRTIM_CHP_PULSEWIDTH_192
  - LL_HRTIM_CHP_PULSEWIDTH_208
  - LL_HRTIM_CHP_PULSEWIDTH_224
  - LL_HRTIM_CHP_PULSEWIDTH_240
  - LL_HRTIM_CHP_PULSEWIDTH_256

Reference Manual to LL API cross reference:

- CHPxR STRPW LL_HRTIM_CHP_GetPulseWidth

LL_HRTIM_OUT_SetOutputSetSrc

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetOutputSetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t SetSrc)

Function description

Set the timer output set source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

- **SetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUTSET_NONE
  - LL_HRTIM_OUTPUTSET_RESYNC
  - LL_HRTIM_OUTPUTSET_TIMPER
  - LL_HRTIM_OUTPUTSET_TIMCMP1
  - LL_HRTIM_OUTPUTSET_TIMCMP2
  - LL_HRTIM_OUTPUTSET_TIMCMP3
  - LL_HRTIM_OUTPUTSET_TIMCMP4
  - LL_HRTIM_OUTPUTSET_MASTERPER
  - LL_HRTIM_OUTPUTSET_MASTERCMP1
  - LL_HRTIM_OUTPUTSET_MASTERCMP2
  - LL_HRTIM_OUTPUTSET_MASTERCMP3
  - LL_HRTIM_OUTPUTSET_MASTERCMP4
  - LL_HRTIM_OUTPUTSET_TIMAEV1_TIMBCMP1
  - LL_HRTIM_OUTPUTSET_TIMAEV2_TIMBCMP2
  - LL_HRTIM_OUTPUTSET_TIMAEV3_TIMFCMP4
  - LL_HRTIM_OUTPUTSET_TIMAEV4_TIMCCMP2
  - LL_HRTIM_OUTPUTSET_TIMAEV5_TIMCCMP3
  - LL_HRTIM_OUTPUTSET_TIMAEV6_TIMDCMP1
  - LL_HRTIM_OUTPUTSET_TIMAEV7_TIMDCMP2
  - LL_HRTIM_OUTPUTSET_TIMAEV8_TIMECMP3
  - LL_HRTIM_OUTPUTSET_TIMAEV9_TIMECMP4
  - LL_HRTIM_OUTPUTSET_TIMBEV1_TIMACMP1
  - LL_HRTIM_OUTPUTSET_TIMBEV2_TIMACMP2
  - LL_HRTIM_OUTPUTSET_TIMBEV3_TIMFCMP3
  - LL_HRTIM_OUTPUTSET_TIMBEV4_TIMCCMP3
  - LL_HRTIM_OUTPUTSET_TIMBEV5_TIMCCMP4
  - LL_HRTIM_OUTPUTSET_TIMBEV6_TIMDCMP3
  - LL_HRTIM_OUTPUTSET_TIMBEV7_TIMDCMP4
  - LL_HRTIM_OUTPUTSET_TIMBEV8_TIMECMP1
  - LL_HRTIM_OUTPUTSET_TIMBEV9_TIMECMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV1_TIMACMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV2_TIMACMP3
  - LL_HRTIM_OUTPUTSET_TIMCEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV4_TIMBCMP3
  - LL_HRTIM_OUTPUTSET_TIMCEV5_TIMDCMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV6_TIMDCMP4
  - LL_HRTIM_OUTPUTSET_TIMCEV7_TIMFCMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV8_TIMECMP3
  - LL_HRTIM_OUTPUTSET_TIMCEV9_TIMECMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV1_TIMACMP1
  - LL_HRTIM_OUTPUTSET_TIMDEV2_TIMACMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTSET_TIMDEV4_TIMBCMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV5_TIMFCMP1
  - LL_HRTIM_OUTPUTSET_TIMDEV6_TIMFCMP3
  - LL_HRTIM_OUTPUTSET_TIMDEV7_TIMCCMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV8_TIMECMP1
  - LL_HRTIM_OUTPUTSET_TIMDEV9_TIMECMP4
  - LL_HRTIM_OUTPUTSET_TIMEEV1_TIMFCMP3
  - LL_HRTIM_OUTPUTSET_TIMEEV2_TIMACMP4
  - LL_HRTIM_OUTPUTSET_TIMEEV3_TIMBCMP3

Return values

- **None:**

Reference Manual to LL API cross reference:

- SETx1R SST LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R SST LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputSetSrc

- SETx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_SetOutputSetSrc

LL_HRTIM_OUT_GetOutputSetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetOutputSetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get the timer output set source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **SetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUTSET_NONE
  - LL_HRTIM_OUTPUTSET_RESYNC
  - LL_HRTIM_OUTPUTSET_TIMPER
  - LL_HRTIM_OUTPUTSET_TIMCMP1
  - LL_HRTIM_OUTPUTSET_TIMCMP2
  - LL_HRTIM_OUTPUTSET_TIMCMP3
  - LL_HRTIM_OUTPUTSET_TIMCMP4
  - LL_HRTIM_OUTPUTSET_MASTERPER
  - LL_HRTIM_OUTPUTSET_MASTERCMP1
  - LL_HRTIM_OUTPUTSET_MASTERCMP2
  - LL_HRTIM_OUTPUTSET_MASTERCMP3
  - LL_HRTIM_OUTPUTSET_MASTERCMP4
  - LL_HRTIM_OUTPUTSET_TIMAEV1_TIMBCMP1
  - LL_HRTIM_OUTPUTSET_TIMAEV2_TIMBCMP2
  - LL_HRTIM_OUTPUTSET_TIMAEV3_TIMFCMP4
  - LL_HRTIM_OUTPUTSET_TIMAEV4_TIMCCMP2
  - LL_HRTIM_OUTPUTSET_TIMAEV5_TIMCCMP3
  - LL_HRTIM_OUTPUTSET_TIMAEV6_TIMDCMP1
  - LL_HRTIM_OUTPUTSET_TIMAEV7_TIMDCMP2
  - LL_HRTIM_OUTPUTSET_TIMAEV8_TIMECMP3
  - LL_HRTIM_OUTPUTSET_TIMAEV9_TIMECMP4
  - LL_HRTIM_OUTPUTSET_TIMBEV1_TIMACMP1
  - LL_HRTIM_OUTPUTSET_TIMBEV2_TIMACMP2
  - LL_HRTIM_OUTPUTSET_TIMBEV3_TIMFCMP3
  - LL_HRTIM_OUTPUTSET_TIMBEV4_TIMCCMP3
  - LL_HRTIM_OUTPUTSET_TIMBEV5_TIMCCMP4
  - LL_HRTIM_OUTPUTSET_TIMBEV6_TIMDCMP3
  - LL_HRTIM_OUTPUTSET_TIMBEV7_TIMDCMP4
  - LL_HRTIM_OUTPUTSET_TIMBEV8_TIMECMP1
  - LL_HRTIM_OUTPUTSET_TIMBEV9_TIMECMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV1_TIMACMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV2_TIMACMP3
  - LL_HRTIM_OUTPUTSET_TIMCEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV4_TIMBCMP3
  - LL_HRTIM_OUTPUTSET_TIMCEV5_TIMDCMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV6_TIMDCMP4
  - LL_HRTIM_OUTPUTSET_TIMCEV7_TIMFCMP2
  - LL_HRTIM_OUTPUTSET_TIMCEV8_TIMECMP3
  - LL_HRTIM_OUTPUTSET_TIMCEV9_TIMECMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV1_TIMACMP1
  - LL_HRTIM_OUTPUTSET_TIMDEV2_TIMACMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTSET_TIMDEV4_TIMBCMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV5_TIMFCMP1
  - LL_HRTIM_OUTPUTSET_TIMDEV6_TIMFCMP3
  - LL_HRTIM_OUTPUTSET_TIMDEV7_TIMCCMP4
  - LL_HRTIM_OUTPUTSET_TIMDEV8_TIMECMP1
  - LL_HRTIM_OUTPUTSET_TIMDEV9_TIMECMP4
  - LL_HRTIM_OUTPUTSET_TIMEEV1_TIMFCMP3
  - LL_HRTIM_OUTPUTSET_TIMEEV2_TIMACMP4

Reference Manual to LL API cross reference:

- SETx1R SST LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R SST LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputSetSrc

- SETx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_GetOutputSetSrc

LL_HRTIM_OUT_SetOutputResetSrc

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetOutputResetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t ResetSrc)

Function description

Set the timer output reset source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

- **ResetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUTRESET_NONE
  - LL_HRTIM_OUTPUTRESET_RESYNC
  - LL_HRTIM_OUTPUTRESET_TIMPER
  - LL_HRTIM_OUTPUTRESET_TIMCMP1
  - LL_HRTIM_OUTPUTRESET_TIMCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCMP3
  - LL_HRTIM_OUTPUTRESET_TIMCMP4
  - LL_HRTIM_OUTPUTRESET_MASTERPER
  - LL_HRTIM_OUTPUTRESET_MASTERCMP1
  - LL_HRTIM_OUTPUTRESET_MASTERCMP2
  - LL_HRTIM_OUTPUTRESET_MASTERCMP3
  - LL_HRTIM_OUTPUTRESET_MASTERCMP4
  - LL_HRTIM_OUTPUTRESET_TIMAEV1_TIMBCMP1
  - LL_HRTIM_OUTPUTRESET_TIMAEV2_TIMBCMP2
  - LL_HRTIM_OUTPUTRESET_TIMAEV3_TIMFCMP4
  - LL_HRTIM_OUTPUTRESET_TIMAEV4_TIMCCMP2
  - LL_HRTIM_OUTPUTRESET_TIMAEV5_TIMCCMP3
  - LL_HRTIM_OUTPUTRESET_TIMAEV6_TIMDCMP1
  - LL_HRTIM_OUTPUTRESET_TIMAEV7_TIMDCMP2
  - LL_HRTIM_OUTPUTRESET_TIMAEV8_TIMECMP3
  - LL_HRTIM_OUTPUTRESET_TIMAEV9_TIMECMP4
  - LL_HRTIM_OUTPUTRESET_TIMBEV1_TIMACMP1
  - LL_HRTIM_OUTPUTRESET_TIMBEV2_TIMACMP2
  - LL_HRTIM_OUTPUTRESET_TIMBEV3_TIMFCMP3
  - LL_HRTIM_OUTPUTRESET_TIMBEV4_TIMCCMP3
  - LL_HRTIM_OUTPUTRESET_TIMBEV5_TIMCCMP4
  - LL_HRTIM_OUTPUTRESET_TIMBEV6_TIMDCMP3
  - LL_HRTIM_OUTPUTRESET_TIMBEV7_TIMDCMP4
  - LL_HRTIM_OUTPUTRESET_TIMBEV8_TIMECMP1
  - LL_HRTIM_OUTPUTRESET_TIMBEV9_TIMECMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV1_TIMACMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV2_TIMACMP3
  - LL_HRTIM_OUTPUTRESET_TIMCEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV4_TIMBCMP3
  - LL_HRTIM_OUTPUTRESET_TIMCEV5_TIMDCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV6_TIMDCMP4
  - LL_HRTIM_OUTPUTRESET_TIMCEV7_TIMFCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV8_TIMECMP3
  - LL_HRTIM_OUTPUTRESET_TIMCEV9_TIMECMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV1_TIMACMP1
  - LL_HRTIM_OUTPUTRESET_TIMDEV2_TIMACMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTRESET_TIMDEV4_TIMBCMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV5_TIMFCMP1
  - LL_HRTIM_OUTPUTRESET_TIMDEV6_TIMFCMP3
  - LL_HRTIM_OUTPUTRESET_TIMDEV7_TIMCCMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV8_TIMECMP1
  - LL_HRTIM_OUTPUTRESET_TIMDEV9_TIMECMP4
  - LL_HRTIM_OUTPUTRESET_TIMEEV1_TIMFCMP3
  - LL_HRTIM_OUTPUTRESET_TIMEEV2_TIMACMP4
  - LL_HRTIM_OUTPUTRESET_TIMEEV3_TIMBCMP3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSTx1R RST LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R RST LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputResetSrc

- RSTx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_SetOutputResetSrc

LL_HRTIM_OUT_GetOutputResetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetOutputResetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get the timer output set source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **ResetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUTRESET_NONE
  - LL_HRTIM_OUTPUTRESET_RESYNC
  - LL_HRTIM_OUTPUTRESET_TIMPER
  - LL_HRTIM_OUTPUTRESET_TIMCMP1
  - LL_HRTIM_OUTPUTRESET_TIMCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCMP3
  - LL_HRTIM_OUTPUTRESET_TIMCMP4
  - LL_HRTIM_OUTPUTRESET_MASTERPER
  - LL_HRTIM_OUTPUTRESET_MASTERCMP1
  - LL_HRTIM_OUTPUTRESET_MASTERCMP2
  - LL_HRTIM_OUTPUTRESET_MASTERCMP3
  - LL_HRTIM_OUTPUTRESET_MASTERCMP4
  - LL_HRTIM_OUTPUTRESET_TIMAEV1_TIMBCMP1
  - LL_HRTIM_OUTPUTRESET_TIMAEV2_TIMBCMP2
  - LL_HRTIM_OUTPUTRESET_TIMAEV3_TIMFCMP4
  - LL_HRTIM_OUTPUTRESET_TIMAEV4_TIMCCMP2
  - LL_HRTIM_OUTPUTRESET_TIMAEV5_TIMCCMP3
  - LL_HRTIM_OUTPUTRESET_TIMAEV6_TIMDCMP1
  - LL_HRTIM_OUTPUTRESET_TIMAEV7_TIMDCMP2
  - LL_HRTIM_OUTPUTRESET_TIMAEV8_TIMECMP3
  - LL_HRTIM_OUTPUTRESET_TIMAEV9_TIMECMP4
  - LL_HRTIM_OUTPUTRESET_TIMBEV1_TIMACMP1
  - LL_HRTIM_OUTPUTRESET_TIMBEV2_TIMACMP2
  - LL_HRTIM_OUTPUTRESET_TIMBEV3_TIMFCMP3
  - LL_HRTIM_OUTPUTRESET_TIMBEV4_TIMCCMP3
  - LL_HRTIM_OUTPUTRESET_TIMBEV5_TIMCCMP4
  - LL_HRTIM_OUTPUTRESET_TIMBEV6_TIMDCMP3
  - LL_HRTIM_OUTPUTRESET_TIMBEV7_TIMDCMP4
  - LL_HRTIM_OUTPUTRESET_TIMBEV8_TIMECMP1
  - LL_HRTIM_OUTPUTRESET_TIMBEV9_TIMECMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV1_TIMACMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV2_TIMACMP3
  - LL_HRTIM_OUTPUTRESET_TIMCEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV4_TIMBCMP3
  - LL_HRTIM_OUTPUTRESET_TIMCEV5_TIMDCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV6_TIMDCMP4
  - LL_HRTIM_OUTPUTRESET_TIMCEV7_TIMFCMP2
  - LL_HRTIM_OUTPUTRESET_TIMCEV8_TIMECMP3
  - LL_HRTIM_OUTPUTRESET_TIMCEV9_TIMECMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV1_TIMACMP1
  - LL_HRTIM_OUTPUTRESET_TIMDEV2_TIMACMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV3_TIMBCMP2
  - LL_HRTIM_OUTPUTRESET_TIMDEV4_TIMBCMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV5_TIMFCMP1
  - LL_HRTIM_OUTPUTRESET_TIMDEV6_TIMFCMP3
  - LL_HRTIM_OUTPUTRESET_TIMDEV7_TIMCCMP4
  - LL_HRTIM_OUTPUTRESET_TIMDEV8_TIMECMP1
  - LL_HRTIM_OUTPUTRESET_TIMDEV9_TIMECMP4
  - LL_HRTIM_OUTPUTRESET_TIMEEV1_TIMFCMP3
  - LL_HRTIM_OUTPUTRESET_TIMEEV2_TIMACMP4

Reference Manual to LL API cross reference:

- RSTx1R RST LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R RST LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputResetSrc

- RSTx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_GetOutputResetSrc

LL_HRTIM_OUT_Config

Function name

__STATIC_INLINE void LL_HRTIM_OUT_Config (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t Configuration)

Function description

Configure a timer output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_OUT_POSITIVE_POLARITY or LL_HRTIM_OUT_NEGATIVE_POLARITY
  - LL_HRTIM_OUT_NO_IDLE or LL_HRTIM_OUT_IDLE_WHEN_BURST
  - LL_HRTIM_OUT_IDLELEVEL_INACTIVE or LL_HRTIM_OUT_IDLELEVEL_ACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_NO_ACTION or LL_HRTIM_OUT_FAULTSTATE_ACTIVE or LL_HRTIM_OUT_FAULTSTATE_INACTIVE or LL_HRTIM_OUT_FAULTSTATE_HIGHZ
  - LL_HRTIM_OUT_CHOPPERMODE_DISABLED or LL_HRTIM_OUT_CHOPPERMODE_ENABLED
  - LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR or LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR POL1 LL_HRTIM_OUT_Config
- OUTxR IDLEM1 LL_HRTIM_OUT_Config
- OUTxR IDLES1 LL_HRTIM_OUT_Config
- OUTxR FAULT1 LL_HRTIM_OUT_Config
- OUTxR CHP1 LL_HRTIM_OUT_Config
- OUTxR DIDL1 LL_HRTIM_OUT_Config
- OUTxR POL2 LL_HRTIM_OUT_Config
- OUTxR IDLEM2 LL_HRTIM_OUT_Config
- OUTxR IDLES2 LL_HRTIM_OUT_Config
- OUTxR FAULT2 LL_HRTIM_OUT_Config
- OUTxR CHP2 LL_HRTIM_OUT_Config
- OUTxR DIDL2 LL_HRTIM_OUT_Config

LL_HRTIM_OUT_SetPolarity

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t Polarity)

Function description

Set the polarity of a timer output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_POSITIVE_POLARITY
  - LL_HRTIM_OUT_NEGATIVE_POLARITY

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR POL1 LL_HRTIM_OUT_SetPolarity
- OUTxR POL2 LL_HRTIM_OUT_SetPolarity

LL_HRTIM_OUT_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual polarity of the timer output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_POSITIVE_POLARITY
  - LL_HRTIM_OUT_NEGATIVE_POLARITY

Reference Manual to LL API cross reference:

- OUTxR POL1 LL_HRTIM_OUT_GetPolarity
- OUTxR POL2 LL_HRTIM_OUT_GetPolarity

LL_HRTIM_OUT_SetIdleMode

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetIdleMode (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t IdleMode)

Function description

Set the output IDLE mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **IdleMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_NO_IDLE
  - LL_HRTIM_OUT_IDLE_WHEN_BURST

Return values

- **None:**

Notes

- This function must not be called when the burst mode is active

Reference Manual to LL API cross reference:

- OUTxR IDLEM1 LL_HRTIM_OUT_SetIdleMode
- OUTxR IDLEM2 LL_HRTIM_OUT_SetIdleMode

LL_HRTIM_OUT_GetIdleMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetIdleMode (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output IDLE mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **IdleMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_NO_IDLE
  - LL_HRTIM_OUT_IDLE_WHEN_BURST

Reference Manual to LL API cross reference:

- OUTxR IDLEM1 LL_HRTIM_OUT_GetIdleMode
- OUTxR IDLEM2 LL_HRTIM_OUT_GetIdleMode

LL_HRTIM_OUT_SetIdleLevel

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetIdleLevel (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t IdleLevel)

Function description

Set the output IDLE level.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **IdleLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_IDLELEVEL_INACTIVE
  - LL_HRTIM_OUT_IDLELEVEL_ACTIVE

Return values

- **None:**

Notes

- This function must be called prior enabling the timer.
- Idle level isn't relevant when the output idle mode is set to LL_HRTIM_OUT_NO_IDLE.

Reference Manual to LL API cross reference:

- OUTxR IDLES1 LL_HRTIM_OUT_SetIdleLevel
- OUTxR IDLES2 LL_HRTIM_OUT_SetIdleLevel

LL_HRTIM_OUT_GetIdleLevel

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetIdleLevel (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output IDLE level.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **IdleLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_IDLELEVEL_INACTIVE
  - LL_HRTIM_OUT_IDLELEVEL_ACTIVE

Reference Manual to LL API cross reference:

- OUTxR IDLES1 LL_HRTIM_OUT_GetIdleLevel
- OUTxR IDLES2 LL_HRTIM_OUT_GetIdleLevel

LL_HRTIM_OUT_SetFaultState

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetFaultState (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t FaultState)

Function description

Set the output FAULT state.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **FaultState:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_FAULTSTATE_NO_ACTION
  - LL_HRTIM_OUT_FAULTSTATE_ACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_INACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_HIGHZ

Return values

- **None:**

Notes

- This function must not called when the timer is enabled and a fault channel is enabled at timer level.

Reference Manual to LL API cross reference:

- OUTxR FAULT1 LL_HRTIM_OUT_SetFaultState
- OUTxR FAULT2 LL_HRTIM_OUT_SetFaultState

LL_HRTIM_OUT_GetFaultState

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetFaultState (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual FAULT state.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **FaultState:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_FAULTSTATE_NO_ACTION
  - LL_HRTIM_OUT_FAULTSTATE_ACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_INACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_HIGHZ

Reference Manual to LL API cross reference:

- OUTxR FAULT1 LL_HRTIM_OUT_GetFaultState
- OUTxR FAULT2 LL_HRTIM_OUT_GetFaultState

LL_HRTIM_OUT_SetChopperMode

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetChopperMode (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t ChopperMode)

Function description

Set the output chopper mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **ChopperMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_CHOPPERMODE_DISABLED
  - LL_HRTIM_OUT_CHOPPERMODE_ENABLED

Return values

- **None:**

Notes

- This function must not called when the timer is enabled.

Reference Manual to LL API cross reference:

- OUTxR CHP1 LL_HRTIM_OUT_SetChopperMode
- OUTxR CHP2 LL_HRTIM_OUT_SetChopperMode

LL_HRTIM_OUT_GetChopperMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetChopperMode (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output chopper mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **ChopperMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_CHOPPERMODE_DISABLED
  - LL_HRTIM_OUT_CHOPPERMODE_ENABLED

Reference Manual to LL API cross reference:

- OUTxR CHP1 LL_HRTIM_OUT_GetChopperMode
- OUTxR CHP2 LL_HRTIM_OUT_GetChopperMode

LL_HRTIM_OUT_SetBMEntryMode

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetBMEntryMode (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t BMEntryMode)

Function description

Set the output burst mode entry mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **BMEntryMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR
  - LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Return values

- **None:**

Notes

- This function must not called when the timer is enabled.

Reference Manual to LL API cross reference:

- OUTxR DIDL1 LL_HRTIM_OUT_SetBMEntryMode
- OUTxR DIDL2 LL_HRTIM_OUT_SetBMEntryMode

LL_HRTIM_OUT_GetBMEntryMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetBMEntryMode (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output burst mode entry mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **BMEntryMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR
  - LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Reference Manual to LL API cross reference:

- OUTxR DIDL1 LL_HRTIM_OUT_GetBMEntryMode
- OUTxR DIDL2 LL_HRTIM_OUT_GetBMEntryMode

LL_HRTIM_OUT_GetDLYPRTOutStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetDLYPRTOutStatus (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get the level (active or inactive) of the designated output when the delayed protection was triggered.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **OutputLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_LEVEL_INACTIVE
  - LL_HRTIM_OUT_LEVEL_ACTIVE

Reference Manual to LL API cross reference:

- TIMxISR O1SRSR LL_HRTIM_OUT_GetDLYPRTOutStatus
- TIMxISR O2SRSR LL_HRTIM_OUT_GetDLYPRTOutStatus

LL_HRTIM_OUT_ForceLevel

Function name

__STATIC_INLINE void LL_HRTIM_OUT_ForceLevel (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t OutputLevel)

Function description

Force the timer output to its active or inactive level.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2
- **OutputLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_LEVEL_INACTIVE
  - LL_HRTIM_OUT_LEVEL_ACTIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- SETx1R SST LL_HRTIM_OUT_ForceLevel
- RSTx1R SRT LL_HRTIM_OUT_ForceLevel
- SETx2R SST LL_HRTIM_OUT_ForceLevel
- RSTx2R SRT LL_HRTIM_OUT_ForceLevel

LL_HRTIM_OUT_GetLevel

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetLevel (HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output level, before the output stage (chopper, polarity).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
  - LL_HRTIM_OUTPUT_TF1
  - LL_HRTIM_OUTPUT_TF2

Return values

- **OutputLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_LEVEL_INACTIVE
  - LL_HRTIM_OUT_LEVEL_ACTIVE

Reference Manual to LL API cross reference:

- TIMxISR O1CPY LL_HRTIM_OUT_GetLevel
- TIMxISR O2CPY LL_HRTIM_OUT_GetLevel

LL_HRTIM_EE_Config

Function name

__STATIC_INLINE void LL_HRTIM_EE_Config (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Configuration)

Function description

Configure external event conditioning.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Configuration:** This parameter must be a combination of all the following values:
  - External event source 1 or External event source 2 or External event source 3 or External event source 4
  - LL_HRTIM_EE_POLARITY_HIGH or LL_HRTIM_EE_POLARITY_LOW
  - LL_HRTIM_EE_SENSITIVITY_LEVEL or LL_HRTIM_EE_SENSITIVITY_RISINGEDGE or LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE or LL_HRTIM_EE_SENSITIVITY_BOTHEDGES
  - LL_HRTIM_EE_FASTMODE_DISABLE or LL_HRTIM_EE_FASTMODE_ENABLE

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.
- Event source (EExSrc1..EExSRC4) mapping depends on configured event channel.
- Fast mode is available only for LL_HRTIM_EVENT_1..5.

Reference Manual to LL API cross reference:

- EECR1 EE1SRC LL_HRTIM_EE_Config
- EECR1 EE1POL LL_HRTIM_EE_Config
- EECR1 EE1SNS LL_HRTIM_EE_Config
- EECR1 EE1FAST LL_HRTIM_EE_Config
- EECR1 EE2SRC LL_HRTIM_EE_Config
- EECR1 EE2POL LL_HRTIM_EE_Config
- EECR1 EE2SNS LL_HRTIM_EE_Config
- EECR1 EE2FAST LL_HRTIM_EE_Config
- EECR1 EE3SRC LL_HRTIM_EE_Config
- EECR1 EE3POL LL_HRTIM_EE_Config
- EECR1 EE3SNS LL_HRTIM_EE_Config
- EECR1 EE3FAST LL_HRTIM_EE_Config
- EECR1 EE4SRC LL_HRTIM_EE_Config
- EECR1 EE4POL LL_HRTIM_EE_Config
- EECR1 EE4SNS LL_HRTIM_EE_Config
- EECR1 EE4FAST LL_HRTIM_EE_Config
- EECR1 EE5SRC LL_HRTIM_EE_Config
- EECR1 EE5POL LL_HRTIM_EE_Config
- EECR1 EE5SNS LL_HRTIM_EE_Config
- EECR1 EE5FAST LL_HRTIM_EE_Config
- EECR2 EE6SRC LL_HRTIM_EE_Config
- EECR2 EE6POL LL_HRTIM_EE_Config
- EECR2 EE6SNS LL_HRTIM_EE_Config
- EECR2 EE6FAST LL_HRTIM_EE_Config
- EECR2 EE7SRC LL_HRTIM_EE_Config
- EECR2 EE7POL LL_HRTIM_EE_Config
- EECR2 EE7SNS LL_HRTIM_EE_Config
- EECR2 EE7FAST LL_HRTIM_EE_Config
- EECR2 EE8SRC LL_HRTIM_EE_Config
- EECR2 EE8POL LL_HRTIM_EE_Config
- EECR2 EE8SNS LL_HRTIM_EE_Config
- EECR2 EE8FAST LL_HRTIM_EE_Config
- EECR2 EE9SRC LL_HRTIM_EE_Config
- EECR2 EE9POL LL_HRTIM_EE_Config
- EECR2 EE9SNS LL_HRTIM_EE_Config
- EECR2 EE9FAST LL_HRTIM_EE_Config
- EECR2 EE10SRC LL_HRTIM_EE_Config
- EECR2 EE10POL LL_HRTIM_EE_Config
- EECR2 EE10SNS LL_HRTIM_EE_Config
- EECR2 EE10FAST LL_HRTIM_EE_Config

LL_HRTIM_EE_SetSrc

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Src)

Function description

Set the external event source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Src:** This parameter can be one of the following values:
  - External event source 1
  - External event source 2
  - External event source 3
  - External event source 4

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR1 EE1SRC LL_HRTIM_EE_SetSrc
- EECR1 EE2SRC LL_HRTIM_EE_SetSrc
- EECR1 EE3SRC LL_HRTIM_EE_SetSrc
- EECR1 EE4SRC LL_HRTIM_EE_SetSrc
- EECR1 EE5SRC LL_HRTIM_EE_SetSrc
- EECR2 EE6SRC LL_HRTIM_EE_SetSrc
- EECR2 EE7SRC LL_HRTIM_EE_SetSrc
- EECR2 EE8SRC LL_HRTIM_EE_SetSrc
- EECR2 EE9SRC LL_HRTIM_EE_SetSrc
- EECR2 EE10SRC LL_HRTIM_EE_SetSrc

LL_HRTIM_EE_GetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual external event source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **EventSrc:** This parameter can be one of the following values:
  - External event source 1
  - External event source 2
  - External event source 3
  - External event source 4

Reference Manual to LL API cross reference:

- EECR1 EE1SRC LL_HRTIM_EE_GetSrc
- EECR1 EE2SRC LL_HRTIM_EE_GetSrc
- EECR1 EE3SRC LL_HRTIM_EE_GetSrc
- EECR1 EE4SRC LL_HRTIM_EE_GetSrc
- EECR1 EE5SRC LL_HRTIM_EE_GetSrc
- EECR2 EE6SRC LL_HRTIM_EE_GetSrc
- EECR2 EE7SRC LL_HRTIM_EE_GetSrc
- EECR2 EE8SRC LL_HRTIM_EE_GetSrc
- EECR2 EE9SRC LL_HRTIM_EE_GetSrc
- EECR2 EE10SRC LL_HRTIM_EE_GetSrc

LL_HRTIM_EE_SetPolarity

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Polarity)

Function description

Set the polarity of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_POLARITY_HIGH
  - LL_HRTIM_EE_POLARITY_LOW

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.
- Event polarity is only significant when event detection is level-sensitive.

Reference Manual to LL API cross reference:

- EECR1 EE1POL LL_HRTIM_EE_SetPolarity
- EECR1 EE2POL LL_HRTIM_EE_SetPolarity
- EECR1 EE3POL LL_HRTIM_EE_SetPolarity
- EECR1 EE4POL LL_HRTIM_EE_SetPolarity
- EECR1 EE5POL LL_HRTIM_EE_SetPolarity
- EECR2 EE6POL LL_HRTIM_EE_SetPolarity
- EECR2 EE7POL LL_HRTIM_EE_SetPolarity
- EECR2 EE8POL LL_HRTIM_EE_SetPolarity
- EECR2 EE9POL LL_HRTIM_EE_SetPolarity
- EECR2 EE10POL LL_HRTIM_EE_SetPolarity

LL_HRTIM_EE_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual polarity setting of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_POLARITY_HIGH
  - LL_HRTIM_EE_POLARITY_LOW

Reference Manual to LL API cross reference:

- EECR1 EE1POL LL_HRTIM_EE_GetPolarity
- EECR1 EE2POL LL_HRTIM_EE_GetPolarity
- EECR1 EE3POL LL_HRTIM_EE_GetPolarity
- EECR1 EE4POL LL_HRTIM_EE_GetPolarity
- EECR1 EE5POL LL_HRTIM_EE_GetPolarity
- EECR2 EE6POL LL_HRTIM_EE_GetPolarity
- EECR2 EE7POL LL_HRTIM_EE_GetPolarity
- EECR2 EE8POL LL_HRTIM_EE_GetPolarity
- EECR2 EE9POL LL_HRTIM_EE_GetPolarity
- EECR2 EE10POL LL_HRTIM_EE_GetPolarity

LL_HRTIM_EE_SetSensitivity

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetSensitivity (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Sensitivity)

Function description

Set the sensitivity of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Sensitivity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_SENSITIVITY_LEVEL
  - LL_HRTIM_EE_SENSITIVITY_RISINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_BOTHEDGES

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR1 EE1SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE2SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE3SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE4SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE5SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE6SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE7SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE8SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE9SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE10SNS LL_HRTIM_EE_SetSensitivity

LL_HRTIM_EE_GetSensitivity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetSensitivity (HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual sensitivity setting of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_SENSITIVITY_LEVEL
  - LL_HRTIM_EE_SENSITIVITY_RISINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_BOTHEDGES

Reference Manual to LL API cross reference:

- EECR1 EE1SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE2SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE3SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE4SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE5SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE6SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE7SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE8SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE9SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE10SNS LL_HRTIM_EE_GetSensitivity

LL_HRTIM_EE_SetFastMode

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetFastMode (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t FastMode)

Function description

Set the fast mode of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
- **FastMode:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FASTMODE_DISABLE
  - LL_HRTIM_EE_FASTMODE_ENABLE

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.

Reference Manual to LL API cross reference:

- EECR1 EE1FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE2FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE3FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE4FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE5FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE6FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE7FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE8FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE9FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE10FAST LL_HRTIM_EE_SetFastMode

LL_HRTIM_EE_GetFastMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetFastMode (HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual fast mode setting of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5

Return values

- **FastMode:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FASTMODE_DISABLE
  - LL_HRTIM_EE_FASTMODE_ENABLE

Reference Manual to LL API cross reference:

- EECR1 EE1FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE2FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE3FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE4FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE5FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE6FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE7FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE8FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE9FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE10FAST LL_HRTIM_EE_GetFastMode

LL_HRTIM_EE_SetFilter

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetFilter (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Filter)

Function description

Set the digital noise filter of a external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FILTER_NONE
  - LL_HRTIM_EE_FILTER_1
  - LL_HRTIM_EE_FILTER_2
  - LL_HRTIM_EE_FILTER_3
  - LL_HRTIM_EE_FILTER_4
  - LL_HRTIM_EE_FILTER_5
  - LL_HRTIM_EE_FILTER_6
  - LL_HRTIM_EE_FILTER_7
  - LL_HRTIM_EE_FILTER_8
  - LL_HRTIM_EE_FILTER_9
  - LL_HRTIM_EE_FILTER_10
  - LL_HRTIM_EE_FILTER_11
  - LL_HRTIM_EE_FILTER_12
  - LL_HRTIM_EE_FILTER_13
  - LL_HRTIM_EE_FILTER_14
  - LL_HRTIM_EE_FILTER_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR3 EE6F LL_HRTIM_EE_SetFilter
- EECR3 EE7F LL_HRTIM_EE_SetFilter
- EECR3 EE8F LL_HRTIM_EE_SetFilter
- EECR3 EE9F LL_HRTIM_EE_SetFilter
- EECR3 EE10F LL_HRTIM_EE_SetFilter

LL_HRTIM_EE_GetFilter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetFilter (HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual digital noise filter setting of a external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FILTER_NONE
  - LL_HRTIM_EE_FILTER_1
  - LL_HRTIM_EE_FILTER_2
  - LL_HRTIM_EE_FILTER_3
  - LL_HRTIM_EE_FILTER_4
  - LL_HRTIM_EE_FILTER_5
  - LL_HRTIM_EE_FILTER_6
  - LL_HRTIM_EE_FILTER_7
  - LL_HRTIM_EE_FILTER_8
  - LL_HRTIM_EE_FILTER_9
  - LL_HRTIM_EE_FILTER_10
  - LL_HRTIM_EE_FILTER_11
  - LL_HRTIM_EE_FILTER_12
  - LL_HRTIM_EE_FILTER_13
  - LL_HRTIM_EE_FILTER_14
  - LL_HRTIM_EE_FILTER_15

Reference Manual to LL API cross reference:

- EECR3 EE6F LL_HRTIM_EE_GetFilter
- EECR3 EE7F LL_HRTIM_EE_GetFilter
- EECR3 EE8F LL_HRTIM_EE_GetFilter
- EECR3 EE9F LL_HRTIM_EE_GetFilter
- EECR3 EE10F LL_HRTIM_EE_GetFilter

LL_HRTIM_EE_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Prescaler)

Function description

Set the external event prescaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_EE_PRESCALER_DIV1
  - LL_HRTIM_EE_PRESCALER_DIV2
  - LL_HRTIM_EE_PRESCALER_DIV4
  - LL_HRTIM_EE_PRESCALER_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR3 EEVSD LL_HRTIM_EE_SetPrescaler

LL_HRTIM_EE_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetPrescaler (HRTIM_TypeDef * HRTIMx)

Function description

Get actual external event prescaler setting.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_EE_PRESCALER_DIV1
  - LL_HRTIM_EE_PRESCALER_DIV2
  - LL_HRTIM_EE_PRESCALER_DIV4
  - LL_HRTIM_EE_PRESCALER_DIV8

Reference Manual to LL API cross reference:

- EECR3 EEVSD LL_HRTIM_EE_GetPrescaler

LL_HRTIM_FLT_Config

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Config (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Configuration)

Function description

Configure fault signal conditioning Polarity and Source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_FLT_SRC_DIGITALINPUT..LL_HRTIM_FLT_SRC_EEVINPUT
  - LL_HRTIM_FLT_POLARITY_LOW..LL_HRTIM_FLT_POLARITY_HIGH

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1P LL_HRTIM_FLT_Config
- FLTINR1 FLT1SRC LL_HRTIM_FLT_Config
- FLTINR1 FLT2P LL_HRTIM_FLT_Config
- FLTINR1 FLT2SRC LL_HRTIM_FLT_Config
- FLTINR1 FLT3P LL_HRTIM_FLT_Config
- FLTINR1 FLT3SRC LL_HRTIM_FLT_Config
- FLTINR1 FLT4P LL_HRTIM_FLT_Config
- FLTINR1 FLT4SRC LL_HRTIM_FLT_Config
- FLTINR2 FLT5P LL_HRTIM_FLT_Config
- FLTINR2 FLT5SRC LL_HRTIM_FLT_Config
- FLTINR2 FLT6P LL_HRTIM_FLT_Config
- FLTINR2 FLT6SRC LL_HRTIM_FLT_Config

LL_HRTIM_FLT_SetSrc

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Src)

Function description

Set the source of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Src:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_SRC_DIGITALINPUT
  - LL_HRTIM_FLT_SRC_INTERNAL
  - LL_HRTIM_FLT_SRC_EEVINPUT

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1SRC LL_HRTIM_FLT_SetSrc
- FLTINR1 FLT2SRC LL_HRTIM_FLT_SetSrc
- FLTINR1 FLT3SRC LL_HRTIM_FLT_SetSrc
- FLTINR1 FLT4SRC LL_HRTIM_FLT_SetSrc
- FLTINR2 FLT5SRC LL_HRTIM_FLT_SetSrc
- FLTINR2 FLT6SRC LL_HRTIM_FLT_SetSrc

LL_HRTIM_FLT_GetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual source of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **Source:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_SRC_DIGITALINPUT
  - LL_HRTIM_FLT_SRC_INTERNAL
  - LL_HRTIM_FLT_SRC_EEVINPUT

Reference Manual to LL API cross reference:

- FLTINR1 FLT1SRC LL_HRTIM_FLT_GetSrc
- FLTINR1 FLT2SRC LL_HRTIM_FLT_GetSrc
- FLTINR1 FLT3SRC LL_HRTIM_FLT_GetSrc
- FLTINR1 FLT4SRC LL_HRTIM_FLT_GetSrc
- FLTINR2 FLT5SRC LL_HRTIM_FLT_GetSrc
- FLTINR2 FLT6SRC LL_HRTIM_FLT_GetSrc

LL_HRTIM_FLT_SetPolarity

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Polarity)

Function description

Set the polarity of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_POLARITY_LOW
  - LL_HRTIM_FLT_POLARITY_HIGH

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1P LL_HRTIM_FLT_SetPolarity
- FLTINR1 FLT2P LL_HRTIM_FLT_SetPolarity
- FLTINR1 FLT3P LL_HRTIM_FLT_SetPolarity
- FLTINR1 FLT4P LL_HRTIM_FLT_SetPolarity
- FLTINR2 FLT5P LL_HRTIM_FLT_SetPolarity
- FLTINR2 FLT6P LL_HRTIM_FLT_SetPolarity

LL_HRTIM_FLT_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual polarity of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_POLARITY_LOW
  - LL_HRTIM_FLT_POLARITY_HIGH

Reference Manual to LL API cross reference:

- FLTINR1 FLT1P LL_HRTIM_FLT_GetPolarity
- FLTINR1 FLT2P LL_HRTIM_FLT_GetPolarity
- FLTINR1 FLT3P LL_HRTIM_FLT_GetPolarity
- FLTINR1 FLT4P LL_HRTIM_FLT_GetPolarity
- FLTINR2 FLT5P LL_HRTIM_FLT_GetPolarity
- FLTINR2 FLT6P LL_HRTIM_FLT_GetPolarity

LL_HRTIM_FLT_SetFilter

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetFilter (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Filter)

Function description

Set the digital noise filter of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_FILTER_NONE
  - LL_HRTIM_FLT_FILTER_1
  - LL_HRTIM_FLT_FILTER_2
  - LL_HRTIM_FLT_FILTER_3
  - LL_HRTIM_FLT_FILTER_4
  - LL_HRTIM_FLT_FILTER_5
  - LL_HRTIM_FLT_FILTER_6
  - LL_HRTIM_FLT_FILTER_7
  - LL_HRTIM_FLT_FILTER_8
  - LL_HRTIM_FLT_FILTER_9
  - LL_HRTIM_FLT_FILTER_10
  - LL_HRTIM_FLT_FILTER_11
  - LL_HRTIM_FLT_FILTER_12
  - LL_HRTIM_FLT_FILTER_13
  - LL_HRTIM_FLT_FILTER_14
  - LL_HRTIM_FLT_FILTER_15

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1F LL_HRTIM_FLT_SetFilter
- FLTINR1 FLT2F LL_HRTIM_FLT_SetFilter
- FLTINR1 FLT3F LL_HRTIM_FLT_SetFilter
- FLTINR1 FLT4F LL_HRTIM_FLT_SetFilter
- FLTINR2 FLT5F LL_HRTIM_FLT_SetFilter
- FLTINR2 FLT6F LL_HRTIM_FLT_SetFilter

LL_HRTIM_FLT_GetFilter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetFilter (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual digital noise filter setting of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_FILTER_NONE
  - LL_HRTIM_FLT_FILTER_1
  - LL_HRTIM_FLT_FILTER_2
  - LL_HRTIM_FLT_FILTER_3
  - LL_HRTIM_FLT_FILTER_4
  - LL_HRTIM_FLT_FILTER_5
  - LL_HRTIM_FLT_FILTER_6
  - LL_HRTIM_FLT_FILTER_7
  - LL_HRTIM_FLT_FILTER_8
  - LL_HRTIM_FLT_FILTER_9
  - LL_HRTIM_FLT_FILTER_10
  - LL_HRTIM_FLT_FILTER_11
  - LL_HRTIM_FLT_FILTER_12
  - LL_HRTIM_FLT_FILTER_13
  - LL_HRTIM_FLT_FILTER_14
  - LL_HRTIM_FLT_FILTER_15

Reference Manual to LL API cross reference:

- FLTINR1 FLT1F LL_HRTIM_FLT_GetFilter
- FLTINR1 FLT2F LL_HRTIM_FLT_GetFilter
- FLTINR1 FLT3F LL_HRTIM_FLT_GetFilter
- FLTINR1 FLT4F LL_HRTIM_FLT_GetFilter
- FLTINR2 FLT5F LL_HRTIM_FLT_GetFilter
- FLTINR2 FLT6F LL_HRTIM_FLT_GetFilter

LL_HRTIM_FLT_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Prescaler)

Function description

Set the fault circuitry prescaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_PRESCALER_DIV1
  - LL_HRTIM_FLT_PRESCALER_DIV2
  - LL_HRTIM_FLT_PRESCALER_DIV4
  - LL_HRTIM_FLT_PRESCALER_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR2 FLTSD LL_HRTIM_FLT_SetPrescaler

LL_HRTIM_FLT_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetPrescaler (HRTIM_TypeDef * HRTIMx)

Function description

Get actual fault circuitry prescaler setting.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_PRESCALER_DIV1
  - LL_HRTIM_FLT_PRESCALER_DIV2
  - LL_HRTIM_FLT_PRESCALER_DIV4
  - LL_HRTIM_FLT_PRESCALER_DIV8

Reference Manual to LL API cross reference:

- FLTINR2 FLTSD LL_HRTIM_FLT_GetPrescaler

LL_HRTIM_FLT_Lock

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Lock (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Lock the fault signal conditioning settings.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1LCK LL_HRTIM_FLT_Lock
- FLTINR1 FLT2LCK LL_HRTIM_FLT_Lock
- FLTINR1 FLT3LCK LL_HRTIM_FLT_Lock
- FLTINR1 FLT4LCK LL_HRTIM_FLT_Lock
- FLTINR2 FLT5LCK LL_HRTIM_FLT_Lock
- FLTINR2 FLT6LCK LL_HRTIM_FLT_Lock

LL_HRTIM_FLT_Enable

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Enable (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Enable the fault circuitry for the designated fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1E LL_HRTIM_FLT_Enable
- FLTINR1 FLT2E LL_HRTIM_FLT_Enable
- FLTINR1 FLT3E LL_HRTIM_FLT_Enable
- FLTINR1 FLT4E LL_HRTIM_FLT_Enable
- FLTINR2 FLT5E LL_HRTIM_FLT_Enable
- FLTINR2 FLT6E LL_HRTIM_FLT_Enable

LL_HRTIM_FLT_Disable

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Disable (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Disable the fault circuitry for for the designated fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1E LL_HRTIM_FLT_Disable
- FLTINR1 FLT2E LL_HRTIM_FLT_Disable
- FLTINR1 FLT3E LL_HRTIM_FLT_Disable
- FLTINR1 FLT4E LL_HRTIM_FLT_Disable
- FLTINR2 FLT5E LL_HRTIM_FLT_Disable
- FLTINR2 FLT6E LL_HRTIM_FLT_Disable

LL_HRTIM_FLT_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_IsEnabled (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Indicate whether the fault circuitry is enabled for a given fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **State:** of FLTxEN bit in HRTIM_FLTINRx register (1 or 0).

Reference Manual to LL API cross reference:

- FLTINR1 FLT1E LL_HRTIM_FLT_IsEnabled
- FLTINR1 FLT2E LL_HRTIM_FLT_IsEnabled
- FLTINR1 FLT3E LL_HRTIM_FLT_IsEnabled
- FLTINR1 FLT4E LL_HRTIM_FLT_IsEnabled
- FLTINR2 FLT5E LL_HRTIM_FLT_IsEnabled
- FLTINR2 FLT6E LL_HRTIM_FLT_IsEnabled

LL_HRTIM_FLT_EnableBlanking

Function name

__STATIC_INLINE void LL_HRTIM_FLT_EnableBlanking (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Enable the Blanking of the fault circuitry for the designated fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1BLKE LL_HRTIM_FLT_EnableBlanking
- FLTINR1 FLT2BLKE LL_HRTIM_FLT_EnableBlanking
- FLTINR1 FLT3BLKE LL_HRTIM_FLT_EnableBlanking
- FLTINR1 FLT4BLKE LL_HRTIM_FLT_EnableBlanking
- FLTINR2 FLT5BLKE LL_HRTIM_FLT_EnableBlanking
- FLTINR2 FLT6BLKE LL_HRTIM_FLT_EnableBlanking

LL_HRTIM_FLT_DisableBlanking

Function name

__STATIC_INLINE void LL_HRTIM_FLT_DisableBlanking (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Disable the Blanking of the fault circuitry for the designated fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1BLKE LL_HRTIM_FLT_DisableBlanking
- FLTINR1 FLT2BLKE LL_HRTIM_FLT_DisableBlanking
- FLTINR1 FLT3BLKE LL_HRTIM_FLT_DisableBlanking
- FLTINR1 FLT4BLKE LL_HRTIM_FLT_DisableBlanking
- FLTINR2 FLT5BLKE LL_HRTIM_FLT_DisableBlanking
- FLTINR2 FLT6BLKE LL_HRTIM_FLT_DisableBlanking

LL_HRTIM_FLT_IsEnabledBlanking

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_IsEnabledBlanking (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Indicate whether the Blanking of the fault circuitry is enabled for a given fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **State:** of FLTxBLKE bit in HRTIM_FLTINRx register (1 or 0).

Reference Manual to LL API cross reference:

- FLTINR1 FLT1BLKE LL_HRTIM_FLT_IsEnabledBlanking
- FLTINR1 FLT2BLKE LL_HRTIM_FLT_IsEnabledBlanking
- FLTINR1 FLT3BLKE LL_HRTIM_FLT_IsEnabledBlanking
- FLTINR1 FLT4BLKE LL_HRTIM_FLT_IsEnabledBlanking
- FLTINR2 FLT5BLKE LL_HRTIM_FLT_IsEnabledBlanking
- FLTINR2 FLT6BLKE LL_HRTIM_FLT_IsEnabledBlanking

LL_HRTIM_FLT_SetBlankingSrc

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetBlankingSrc (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Source)

Function description

Set the Blanking Source of the fault circuitry for a given fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Source:** parameter can be one of the following values:
  - LL_HRTIM_FLT_BLANKING_RSTALIGNED
  - LL_HRTIM_FLT_BLANKING_MOVING

Return values

- **None:**

Notes

- Fault inputs can be temporary disabled to blank spurious fault events.
- This function allows for selection amongst 2 possible blanking sources.
- Events triggering blanking window start and blanking window end depend on both the selected blanking source and the fault input.

Reference Manual to LL API cross reference:

- FLTINR3 FLT1BLKS LL_HRTIM_FLT_SetBlankingSrc
- FLTINR3 FLT2BLKS LL_HRTIM_FLT_SetBlankingSrc
- FLTINR3 FLT3BLKS LL_HRTIM_FLT_SetBlankingSrc
- FLTINR3 FLT4BLKS LL_HRTIM_FLT_SetBlankingSrc
- FLTINR4 FLT5BLKS LL_HRTIM_FLT_SetBlankingSrc
- FLTINR4 FLT6BLKS LL_HRTIM_FLT_SetBlankingSrc

LL_HRTIM_FLT_GetBlankingSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetBlankingSrc (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get the Blanking Source of the fault circuitry is enabled for a given fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Reference Manual to LL API cross reference:

- FLTINR3 FLT1BLKS LL_HRTIM_FLT_GetBlankingSrc
- FLTINR3 FLT2BLKS LL_HRTIM_FLT_GetBlankingSrc
- FLTINR3 FLT3BLKS LL_HRTIM_FLT_GetBlankingSrc
- FLTINR3 FLT4BLKS LL_HRTIM_FLT_GetBlankingSrc
- FLTINR4 FLT5BLKS LL_HRTIM_FLT_GetBlankingSrc
- FLTINR4 FLT6BLKS LL_HRTIM_FLT_GetBlankingSrc

LL_HRTIM_FLT_SetCounterThreshold

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetCounterThreshold (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Threshold)

Function description

Set the Counter threshold value of a fault counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Threshold:** This parameter can be a number between Min_Data=0 and Max_Data=15

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR3 FLT1CNT LL_HRTIM_FLT_SetCounterThreshold
- FLTINR3 FLT2CNT LL_HRTIM_FLT_SetCounterThreshold
- FLTINR3 FLT3CNT LL_HRTIM_FLT_SetCounterThreshold
- FLTINR3 FLT4CNT LL_HRTIM_FLT_SetCounterThreshold
- FLTINR4 FLT5CNT LL_HRTIM_FLT_SetCounterThreshold
- FLTINR4 FLT6CNT LL_HRTIM_FLT_SetCounterThreshold

LL_HRTIM_FLT_GetCounterThreshold

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetCounterThreshold (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual the Counter threshold value of a fault counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **Threshold:** This parameter can be a number between Min_Data=0 and Max_Data=15

Reference Manual to LL API cross reference:

- FLTINR3 FLT1CNT LL_HRTIM_FLT_GetCounterThreshold
- FLTINR3 FLT2CNT LL_HRTIM_FLT_GetCounterThreshold
- FLTINR3 FLT3CNT LL_HRTIM_FLT_GetCounterThreshold
- FLTINR3 FLT4CNT LL_HRTIM_FLT_GetCounterThreshold
- FLTINR4 FLT5CNT LL_HRTIM_FLT_GetCounterThreshold
- FLTINR4 FLT6CNT LL_HRTIM_FLT_GetCounterThreshold

LL_HRTIM_FLT_SetResetMode

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetResetMode (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Mode)

Function description

Set the mode of reset of a fault counter to 'always reset'.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_COUNTERRST_UNCONDITIONAL
  - LL_HRTIM_FLT_COUNTERRST_CONDITIONAL

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR3 FLT1RSTM LL_HRTIM_FLT_SetResetMode
- FLTINR3 FLT2RSTM LL_HRTIM_FLT_SetResetMode
- FLTINR3 FLT3RSTM LL_HRTIM_FLT_SetResetMode
- FLTINR3 FLT4RSTM LL_HRTIM_FLT_SetResetMode
- FLTINR4 FLT5RSTM LL_HRTIM_FLT_SetResetMode
- FLTINR4 FLT6RSTM LL_HRTIM_FLT_SetResetMode

LL_HRTIM_FLT_GetResetMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetResetMode (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get the mode of reset of a fault counter to 'reset on event'.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_COUNTERRST_UNCONDITIONAL
  - LL_HRTIM_FLT_COUNTERRST_CONDITIONAL

Reference Manual to LL API cross reference:

- FLTINR3 FLT1RSTM LL_HRTIM_FLT_GetResetMode
- FLTINR3 FLT2RSTM LL_HRTIM_FLT_GetResetMode
- FLTINR3 FLT3RSTM LL_HRTIM_FLT_GetResetMode
- FLTINR3 FLT4RSTM LL_HRTIM_FLT_GetResetMode
- FLTINR4 FLT5RSTM LL_HRTIM_FLT_GetResetMode
- FLTINR4 FLT6RSTM LL_HRTIM_FLT_GetResetMode

LL_HRTIM_FLT_ResetCounter

Function name

__STATIC_INLINE void LL_HRTIM_FLT_ResetCounter (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Reset the fault counter for a fault circuitry.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
  - LL_HRTIM_FAULT_6

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR3 FLT1RSTM LL_HRTIM_FLT_ResetCounter
- FLTINR3 FLT2RSTM LL_HRTIM_FLT_ResetCounter
- FLTINR3 FLT3RSTM LL_HRTIM_FLT_ResetCounter
- FLTINR3 FLT4RSTM LL_HRTIM_FLT_ResetCounter
- FLTINR4 FLT5RSTM LL_HRTIM_FLT_ResetCounter
- FLTINR4 FLT6RSTM LL_HRTIM_FLT_ResetCounter

LL_HRTIM_BM_Config

Function name

__STATIC_INLINE void LL_HRTIM_BM_Config (HRTIM_TypeDef * HRTIMx, uint32_t Configuration)

Function description

Configure the burst mode controller.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_BM_MODE_SINGLESHOT or LL_HRTIM_BM_MODE_CONTINOUS
  - LL_HRTIM_BM_CLKSRC_MASTER or ... or LL_HRTIM_BM_CLKSRC_FHRTIM
  - LL_HRTIM_BM_PRESCALER_DIV1 or ... LL_HRTIM_BM_PRESCALER_DIV32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMOM LL_HRTIM_BM_Config
- BMCR BMCLK LL_HRTIM_BM_Config
- BMCR BMPRSC LL_HRTIM_BM_Config

LL_HRTIM_BM_SetMode

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetMode (HRTIM_TypeDef * HRTIMx, uint32_t Mode)

Function description

Set the burst mode controller operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_BM_MODE_SINGLESHOT
  - LL_HRTIM_BM_MODE_CONTINOUS

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMOM LL_HRTIM_BM_SetMode

LL_HRTIM_BM_GetMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetMode (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_BM_MODE_SINGLESHOT
  - LL_HRTIM_BM_MODE_CONTINOUS

Reference Manual to LL API cross reference:

- BMCR BMOM LL_HRTIM_BM_GetMode

LL_HRTIM_BM_SetClockSrc

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetClockSrc (HRTIM_TypeDef * HRTIMx, uint32_t ClockSrc)

Function description

Set the burst mode controller clock source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ClockSrc:** This parameter can be one of the following values:
  - LL_HRTIM_BM_CLKSRC_MASTER
  - LL_HRTIM_BM_CLKSRC_TIMER_A
  - LL_HRTIM_BM_CLKSRC_TIMER_B
  - LL_HRTIM_BM_CLKSRC_TIMER_C
  - LL_HRTIM_BM_CLKSRC_TIMER_D
  - LL_HRTIM_BM_CLKSRC_TIMER_E
  - LL_HRTIM_BM_CLKSRC_TIM16_OC
  - LL_HRTIM_BM_CLKSRC_TIM17_OC
  - LL_HRTIM_BM_CLKSRC_TIM7_TRGO
  - LL_HRTIM_BM_CLKSRC_FHRTIM
  - LL_HRTIM_BM_CLKSRC_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMCLK LL_HRTIM_BM_SetClockSrc

LL_HRTIM_BM_GetClockSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetClockSrc (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller clock source.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **ClockSrc:** This parameter can be one of the following values:
  - LL_HRTIM_BM_CLKSRC_MASTER
  - LL_HRTIM_BM_CLKSRC_TIMER_A
  - LL_HRTIM_BM_CLKSRC_TIMER_B
  - LL_HRTIM_BM_CLKSRC_TIMER_C
  - LL_HRTIM_BM_CLKSRC_TIMER_D
  - LL_HRTIM_BM_CLKSRC_TIMER_E
  - LL_HRTIM_BM_CLKSRC_TIM16_OC
  - LL_HRTIM_BM_CLKSRC_TIM17_OC
  - LL_HRTIM_BM_CLKSRC_TIM7_TRGO
  - LL_HRTIM_BM_CLKSRC_FHRTIM
  - LL_HRTIM_BM_CLKSRC_TIMER_F
- **ClockSrc:** This parameter can be one of the following values:
  - LL_HRTIM_BM_CLKSRC_MASTER
  - LL_HRTIM_BM_CLKSRC_TIMER_A
  - LL_HRTIM_BM_CLKSRC_TIMER_B
  - LL_HRTIM_BM_CLKSRC_TIMER_C
  - LL_HRTIM_BM_CLKSRC_TIMER_D
  - LL_HRTIM_BM_CLKSRC_TIMER_E
  - LL_HRTIM_BM_CLKSRC_TIM16_OC
  - LL_HRTIM_BM_CLKSRC_TIM17_OC
  - LL_HRTIM_BM_CLKSRC_TIM7_TRGO
  - LL_HRTIM_BM_CLKSRC_FHRTIM

Reference Manual to LL API cross reference:

- BMCR BMCLK LL_HRTIM_BM_GetClockSrc

LL_HRTIM_BM_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Prescaler)

Function description

Set the burst mode controller prescaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_BM_PRESCALER_DIV1
  - LL_HRTIM_BM_PRESCALER_DIV2
  - LL_HRTIM_BM_PRESCALER_DIV4
  - LL_HRTIM_BM_PRESCALER_DIV8
  - LL_HRTIM_BM_PRESCALER_DIV16
  - LL_HRTIM_BM_PRESCALER_DIV32
  - LL_HRTIM_BM_PRESCALER_DIV64
  - LL_HRTIM_BM_PRESCALER_DIV128
  - LL_HRTIM_BM_PRESCALER_DIV256
  - LL_HRTIM_BM_PRESCALER_DIV512
  - LL_HRTIM_BM_PRESCALER_DIV1024
  - LL_HRTIM_BM_PRESCALER_DIV2048
  - LL_HRTIM_BM_PRESCALER_DIV4096
  - LL_HRTIM_BM_PRESCALER_DIV8192
  - LL_HRTIM_BM_PRESCALER_DIV16384
  - LL_HRTIM_BM_PRESCALER_DIV32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMPRSC LL_HRTIM_BM_SetPrescaler

LL_HRTIM_BM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetPrescaler (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller prescaler setting.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_BM_PRESCALER_DIV1
  - LL_HRTIM_BM_PRESCALER_DIV2
  - LL_HRTIM_BM_PRESCALER_DIV4
  - LL_HRTIM_BM_PRESCALER_DIV8
  - LL_HRTIM_BM_PRESCALER_DIV16
  - LL_HRTIM_BM_PRESCALER_DIV32
  - LL_HRTIM_BM_PRESCALER_DIV64
  - LL_HRTIM_BM_PRESCALER_DIV128
  - LL_HRTIM_BM_PRESCALER_DIV256
  - LL_HRTIM_BM_PRESCALER_DIV512
  - LL_HRTIM_BM_PRESCALER_DIV1024
  - LL_HRTIM_BM_PRESCALER_DIV2048
  - LL_HRTIM_BM_PRESCALER_DIV4096
  - LL_HRTIM_BM_PRESCALER_DIV8192
  - LL_HRTIM_BM_PRESCALER_DIV16384
  - LL_HRTIM_BM_PRESCALER_DIV32768

Reference Manual to LL API cross reference:

- BMCR BMPRSC LL_HRTIM_BM_GetPrescaler

LL_HRTIM_BM_EnablePreload

Function name

__STATIC_INLINE void LL_HRTIM_BM_EnablePreload (HRTIM_TypeDef * HRTIMx)

Function description

Enable burst mode compare and period registers preload.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMPREN LL_HRTIM_BM_EnablePreload

LL_HRTIM_BM_DisablePreload

Function name

__STATIC_INLINE void LL_HRTIM_BM_DisablePreload (HRTIM_TypeDef * HRTIMx)

Function description

Disable burst mode compare and period registers preload.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMPREN LL_HRTIM_BM_DisablePreload

LL_HRTIM_BM_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_IsEnabledPreload (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether burst mode compare and period registers are preloaded.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BMPREN bit in HRTIM_BMCR register (1 or 0).

Reference Manual to LL API cross reference:

- BMCR BMPREN LL_HRTIM_BM_IsEnabledPreload

LL_HRTIM_BM_SetTrig

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetTrig (HRTIM_TypeDef * HRTIMx, uint32_t Trig)

Function description

Set the burst mode controller trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Trig:** This parameter can be a combination of the following values:
  - LL_HRTIM_BM_TRIG_NONE
  - LL_HRTIM_BM_TRIG_MASTER_RESET
  - LL_HRTIM_BM_TRIG_MASTER_REPETITION
  - LL_HRTIM_BM_TRIG_MASTER_CMP1
  - LL_HRTIM_BM_TRIG_MASTER_CMP2
  - LL_HRTIM_BM_TRIG_MASTER_CMP3
  - LL_HRTIM_BM_TRIG_MASTER_CMP4
  - LL_HRTIM_BM_TRIG_TIMA_RESET
  - LL_HRTIM_BM_TRIG_TIMA_REPETITION
  - LL_HRTIM_BM_TRIG_TIMA_CMP1
  - LL_HRTIM_BM_TRIG_TIMA_CMP2
  - LL_HRTIM_BM_TRIG_TIMB_RESET
  - LL_HRTIM_BM_TRIG_TIMB_REPETITION
  - LL_HRTIM_BM_TRIG_TIMB_CMP1
  - LL_HRTIM_BM_TRIG_TIMB_CMP2
  - LL_HRTIM_BM_TRIG_TIMC_RESET
  - LL_HRTIM_BM_TRIG_TIMC_REPETITION
  - LL_HRTIM_BM_TRIG_TIMC_CMP1
  - LL_HRTIM_BM_TRIG_TIMD_RESET
  - LL_HRTIM_BM_TRIG_TIMD_REPETITION
  - LL_HRTIM_BM_TRIG_TIMD_CMP2
  - LL_HRTIM_BM_TRIG_TIME_REPETITION
  - LL_HRTIM_BM_TRIG_TIME_CMP1
  - LL_HRTIM_BM_TRIG_TIME_CMP2
  - LL_HRTIM_BM_TRIG_TIMF_RESET
  - LL_HRTIM_BM_TRIG_TIMF_REPETITION
  - LL_HRTIM_BM_TRIG_TIMF_CMP1
  - LL_HRTIM_BM_TRIG_TIMA_EVENT7
  - LL_HRTIM_BM_TRIG_TIMD_EVENT8
  - LL_HRTIM_BM_TRIG_EVENT_7
  - LL_HRTIM_BM_TRIG_EVENT_8
  - LL_HRTIM_BM_TRIG_EVENT_ONCHIP

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMTRGR SW LL_HRTIM_BM_SetTrig
- BMTRGR MSTRST LL_HRTIM_BM_SetTrig
- BMTRGR MSTREP LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP3 LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP4 LL_HRTIM_BM_SetTrig
- BMTRGR TARST LL_HRTIM_BM_SetTrig
- BMTRGR TAREP LL_HRTIM_BM_SetTrig
- BMTRGR TACMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TACMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TBRST LL_HRTIM_BM_SetTrig
- BMTRGR TBREP LL_HRTIM_BM_SetTrig
- BMTRGR TBCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TBCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TCRST LL_HRTIM_BM_SetTrig
- BMTRGR TCREP LL_HRTIM_BM_SetTrig
- BMTRGR TCCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TDRST LL_HRTIM_BM_SetTrig
- BMTRGR TDREP LL_HRTIM_BM_SetTrig
- BMTRGR TDCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TEREP LL_HRTIM_BM_SetTrig
- BMTRGR TECMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TECMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TFREP LL_HRTIM_BM_SetTrig
- BMTRGR TFRST LL_HRTIM_BM_SetTrig
- BMTRGR TFCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TAEEV7 LL_HRTIM_BM_SetTrig
- BMTRGR TAEEV8 LL_HRTIM_BM_SetTrig
- BMTRGR EEV7 LL_HRTIM_BM_SetTrig
- BMTRGR EEV8 LL_HRTIM_BM_SetTrig
- BMTRGR OCHIPEV LL_HRTIM_BM_SetTrig

LL_HRTIM_BM_GetTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetTrig (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Trig:** This parameter can be a combination of the following values:
  - LL_HRTIM_BM_TRIG_NONE
  - LL_HRTIM_BM_TRIG_MASTER_RESET
  - LL_HRTIM_BM_TRIG_MASTER_REPETITION
  - LL_HRTIM_BM_TRIG_MASTER_CMP1
  - LL_HRTIM_BM_TRIG_MASTER_CMP2
  - LL_HRTIM_BM_TRIG_MASTER_CMP3
  - LL_HRTIM_BM_TRIG_MASTER_CMP4
  - LL_HRTIM_BM_TRIG_TIMA_RESET
  - LL_HRTIM_BM_TRIG_TIMA_REPETITION
  - LL_HRTIM_BM_TRIG_TIMA_CMP1
  - LL_HRTIM_BM_TRIG_TIMA_CMP2
  - LL_HRTIM_BM_TRIG_TIMB_RESET
  - LL_HRTIM_BM_TRIG_TIMB_REPETITION
  - LL_HRTIM_BM_TRIG_TIMB_CMP1
  - LL_HRTIM_BM_TRIG_TIMB_CMP2
  - LL_HRTIM_BM_TRIG_TIMC_RESET
  - LL_HRTIM_BM_TRIG_TIMC_REPETITION
  - LL_HRTIM_BM_TRIG_TIMC_CMP1
  - LL_HRTIM_BM_TRIG_TIMD_RESET
  - LL_HRTIM_BM_TRIG_TIMD_REPETITION
  - LL_HRTIM_BM_TRIG_TIMD_CMP2
  - LL_HRTIM_BM_TRIG_TIME_REPETITION
  - LL_HRTIM_BM_TRIG_TIME_CMP1
  - LL_HRTIM_BM_TRIG_TIME_CMP2
  - LL_HRTIM_BM_TRIG_TIMF_RESET
  - LL_HRTIM_BM_TRIG_TIMF_REPETITION
  - LL_HRTIM_BM_TRIG_TIMF_CMP1
  - LL_HRTIM_BM_TRIG_TIMA_EVENT7
  - LL_HRTIM_BM_TRIG_TIMD_EVENT8
  - LL_HRTIM_BM_TRIG_EVENT_7
  - LL_HRTIM_BM_TRIG_EVENT_8
  - LL_HRTIM_BM_TRIG_EVENT_ONCHIP

Reference Manual to LL API cross reference:

- BMTRGR SW LL_HRTIM_BM_GetTrig
- BMTRGR MSTRST LL_HRTIM_BM_GetTrig
- BMTRGR MSTREP LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP3 LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP4 LL_HRTIM_BM_GetTrig
- BMTRGR TARST LL_HRTIM_BM_GetTrig
- BMTRGR TAREP LL_HRTIM_BM_GetTrig
- BMTRGR TACMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TACMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TBRST LL_HRTIM_BM_GetTrig
- BMTRGR TBREP LL_HRTIM_BM_GetTrig
- BMTRGR TBCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TBCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TCRST LL_HRTIM_BM_GetTrig
- BMTRGR TCREP LL_HRTIM_BM_GetTrig
- BMTRGR TCCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TDRST LL_HRTIM_BM_GetTrig
- BMTRGR TDREP LL_HRTIM_BM_GetTrig
- BMTRGR TDCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TEREP LL_HRTIM_BM_GetTrig
- BMTRGR TECMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TECMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TFREP LL_HRTIM_BM_GetTrig
- BMTRGR TFRST LL_HRTIM_BM_GetTrig
- BMTRGR TFCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TAEEV7 LL_HRTIM_BM_GetTrig
- BMTRGR TAEEV8 LL_HRTIM_BM_GetTrig
- BMTRGR EEV7 LL_HRTIM_BM_GetTrig
- BMTRGR EEV8 LL_HRTIM_BM_GetTrig
- BMTRGR OCHIPEV LL_HRTIM_BM_GetTrig

LL_HRTIM_BM_SetCompare

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetCompare (HRTIM_TypeDef * HRTIMx, uint32_t CompareValue)

Function description

Set the burst mode controller compare value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCMPR BMCMP LL_HRTIM_BM_SetCompare

LL_HRTIM_BM_GetCompare

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetCompare (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller compare value.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- BMCMPR BMCMP LL_HRTIM_BM_GetCompare

LL_HRTIM_BM_SetPeriod

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetPeriod (HRTIM_TypeDef * HRTIMx, uint32_t Period)

Function description

Set the burst mode controller period.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Period:** The period value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,... The maximum value is 0x0000 FFDF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMPER BMPER LL_HRTIM_BM_SetPeriod

LL_HRTIM_BM_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetPeriod (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller period.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **The:** period value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,... The maximum value is 0x0000 FFDF.

Reference Manual to LL API cross reference:

- BMPER BMPER LL_HRTIM_BM_GetPeriod

LL_HRTIM_BM_Enable

Function name

__STATIC_INLINE void LL_HRTIM_BM_Enable (HRTIM_TypeDef * HRTIMx)

Function description

Enable the burst mode controller.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BME LL_HRTIM_BM_Enable

LL_HRTIM_BM_Disable

Function name

__STATIC_INLINE void LL_HRTIM_BM_Disable (HRTIM_TypeDef * HRTIMx)

Function description

Disable the burst mode controller.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BME LL_HRTIM_BM_Disable

LL_HRTIM_BM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_IsEnabled (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the burst mode controller is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BME bit in HRTIM_BMCR register (1 or 0).

Reference Manual to LL API cross reference:

- BMCR BME LL_HRTIM_BM_IsEnabled

LL_HRTIM_BM_Start

Function name

__STATIC_INLINE void LL_HRTIM_BM_Start (HRTIM_TypeDef * HRTIMx)

Function description

Trigger the burst operation (software trigger)

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMTRGR SW LL_HRTIM_BM_Start

LL_HRTIM_BM_Stop

Function name

__STATIC_INLINE void LL_HRTIM_BM_Stop (HRTIM_TypeDef * HRTIMx)

Function description

Stop the burst mode operation.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Notes

- Causes a burst mode early termination.

Reference Manual to LL API cross reference:

- BMCR BMSTAT LL_HRTIM_BM_Stop

LL_HRTIM_BM_GetStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetStatus (HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode status.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Status:** This parameter can be one of the following values:
  - LL_HRTIM_BM_STATUS_NORMAL
  - LL_HRTIM_BM_STATUS_BURST_ONGOING

Reference Manual to LL API cross reference:

- BMCR BMSTAT LL_HRTIM_BM_GetStatus

LL_HRTIM_ClearFlag_FLT1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 1 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT1C LL_HRTIM_ClearFlag_FLT1

LL_HRTIM_IsActiveFlag_FLT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 1 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT1 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT1 LL_HRTIM_IsActiveFlag_FLT1

LL_HRTIM_ClearFlag_FLT2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 2 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT2C LL_HRTIM_ClearFlag_FLT2

LL_HRTIM_IsActiveFlag_FLT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 2 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT2 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT2 LL_HRTIM_IsActiveFlag_FLT2

LL_HRTIM_ClearFlag_FLT3

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 3 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT3C LL_HRTIM_ClearFlag_FLT3

LL_HRTIM_IsActiveFlag_FLT3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 3 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT3 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT3 LL_HRTIM_IsActiveFlag_FLT3

LL_HRTIM_ClearFlag_FLT4

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 4 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT4C LL_HRTIM_ClearFlag_FLT4

LL_HRTIM_IsActiveFlag_FLT4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 4 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT4 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT4 LL_HRTIM_IsActiveFlag_FLT4

LL_HRTIM_ClearFlag_FLT5

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 5 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT5C LL_HRTIM_ClearFlag_FLT5

LL_HRTIM_IsActiveFlag_FLT5

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 5 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT5 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT5 LL_HRTIM_IsActiveFlag_FLT5

LL_HRTIM_ClearFlag_FLT6

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT6 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 6 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT6C LL_HRTIM_ClearFlag_FLT6

LL_HRTIM_IsActiveFlag_FLT6

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT6 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 6 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT6 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT6 LL_HRTIM_IsActiveFlag_FLT6

LL_HRTIM_ClearFlag_SYSFLT

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Clear the System Fault interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYSFLTC LL_HRTIM_ClearFlag_SYSFLT

LL_HRTIM_IsActiveFlag_SYSFLT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether System Fault interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYSFLT bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYSFLT LL_HRTIM_IsActiveFlag_SYSFLT

LL_HRTIM_ClearFlag_DLLRDY

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_DLLRDY (HRTIM_TypeDef * HRTIMx)

Function description

Clear the DLL ready interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DLLRDYC LL_HRTIM_ClearFlag_DLLRDY

LL_HRTIM_IsActiveFlag_DLLRDY

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_DLLRDY (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether DLL ready interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of DLLRDY bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ISR DLLRDY LL_HRTIM_IsActiveFlag_DLLRDY

LL_HRTIM_ClearFlag_BMPER

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Burst Mode period interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BMPERC LL_HRTIM_ClearFlag_BMPER

LL_HRTIM_IsActiveFlag_BMPER

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Burst Mode period interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BMPER bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ISR BMPER LL_HRTIM_IsActiveFlag_BMPER

LL_HRTIM_ClearFlag_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Synchronization Input interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR SYNCC LL_HRTIM_ClearFlag_SYNC

LL_HRTIM_IsActiveFlag_SYNC

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the Synchronization Input interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYNC bit in HRTIM_MISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR SYNC LL_HRTIM_IsActiveFlag_SYNC

LL_HRTIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the update interrupt flag for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MUPDC LL_HRTIM_ClearFlag_UPDATE
- TIMxICR UPDC LL_HRTIM_ClearFlag_UPDATE

LL_HRTIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the update interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MUPD/UPD bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MUPD LL_HRTIM_IsActiveFlag_UPDATE
- TIMxISR UPD LL_HRTIM_IsActiveFlag_UPDATE

LL_HRTIM_ClearFlag_REP

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the repetition interrupt flag for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MREPC LL_HRTIM_ClearFlag_REP
- TIMxICR REPC LL_HRTIM_ClearFlag_REP

LL_HRTIM_IsActiveFlag_REP

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the repetition interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MREP/REP bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MREP LL_HRTIM_IsActiveFlag_REP
- TIMxISR REP LL_HRTIM_IsActiveFlag_REP

LL_HRTIM_ClearFlag_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 1 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP1C LL_HRTIM_ClearFlag_CMP1
- TIMxICR CMP1C LL_HRTIM_ClearFlag_CMP1

LL_HRTIM_IsActiveFlag_CMP1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 1 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP1/CMP1 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP1 LL_HRTIM_IsActiveFlag_CMP1
- TIMxISR CMP1 LL_HRTIM_IsActiveFlag_CMP1

LL_HRTIM_ClearFlag_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 2 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP2C LL_HRTIM_ClearFlag_CMP2
- TIMxICR CMP2C LL_HRTIM_ClearFlag_CMP2

LL_HRTIM_IsActiveFlag_CMP2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 2 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP2/CMP2 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP2 LL_HRTIM_IsActiveFlag_CMP2
- TIMxISR CMP2 LL_HRTIM_IsActiveFlag_CMP2

LL_HRTIM_ClearFlag_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 3 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP3C LL_HRTIM_ClearFlag_CMP3
- TIMxICR CMP3C LL_HRTIM_ClearFlag_CMP3

LL_HRTIM_IsActiveFlag_CMP3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 3 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP3/CMP3 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP3 LL_HRTIM_IsActiveFlag_CMP3
- TIMxISR CMP3 LL_HRTIM_IsActiveFlag_CMP3

LL_HRTIM_ClearFlag_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 4 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP4C LL_HRTIM_ClearFlag_CMP4
- TIMxICR CMP4C LL_HRTIM_ClearFlag_CMP4

LL_HRTIM_IsActiveFlag_CMP4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 4 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP4/CMP4 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP4 LL_HRTIM_IsActiveFlag_CMP4
- TIMxISR CMP4 LL_HRTIM_IsActiveFlag_CMP4

LL_HRTIM_ClearFlag_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the capture 1 interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR CPT1C LL_HRTIM_ClearFlag_CPT1

LL_HRTIM_IsActiveFlag_CPT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 1 interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of CPT1 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR CPT1 LL_HRTIM_IsActiveFlag_CPT1

LL_HRTIM_ClearFlag_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the capture 2 interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR CPT2C LL_HRTIM_ClearFlag_CPT2

LL_HRTIM_IsActiveFlag_CPT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 2 interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of CPT2 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR CPT2 LL_HRTIM_IsActiveFlag_CPT2

LL_HRTIM_ClearFlag_SET1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 1 set interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR SET1C LL_HRTIM_ClearFlag_SET1

LL_HRTIM_IsActiveFlag_SET1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 set interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SETx1 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR SET1 LL_HRTIM_IsActiveFlag_SET1

LL_HRTIM_ClearFlag_RST1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 1 reset interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR RST1C LL_HRTIM_ClearFlag_RST1

LL_HRTIM_IsActiveFlag_RST1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 reset interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RSTx1 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR RST1 LL_HRTIM_IsActiveFlag_RST1

LL_HRTIM_ClearFlag_SET2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 2 set interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR SET2C LL_HRTIM_ClearFlag_SET2

LL_HRTIM_IsActiveFlag_SET2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 set interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SETx2 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR SET2 LL_HRTIM_IsActiveFlag_SET2

LL_HRTIM_ClearFlag_RST2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 2reset interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR RST2C LL_HRTIM_ClearFlag_RST2

LL_HRTIM_IsActiveFlag_RST2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 reset interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RSTx2 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR RST2 LL_HRTIM_IsActiveFlag_RST2

LL_HRTIM_ClearFlag_RST

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the reset and/or roll-over interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR RSTC LL_HRTIM_ClearFlag_RST

LL_HRTIM_IsActiveFlag_RST

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the reset and/or roll-over interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RST bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR RST LL_HRTIM_IsActiveFlag_RST

LL_HRTIM_ClearFlag_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the delayed protection interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR DLYPRTC LL_HRTIM_ClearFlag_DLYPRT

LL_HRTIM_IsActiveFlag_DLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the delayed protection interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DLYPRT bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR DLYPRT LL_HRTIM_IsActiveFlag_DLYPRT

LL_HRTIM_EnableIT_FLT1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 1 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT1IE LL_HRTIM_EnableIT_FLT1

LL_HRTIM_DisableIT_FLT1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 1 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT1IE LL_HRTIM_DisableIT_FLT1

LL_HRTIM_IsEnabledIT_FLT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 1 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT1IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT1IE LL_HRTIM_IsEnabledIT_FLT1

LL_HRTIM_EnableIT_FLT2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 2 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT2IE LL_HRTIM_EnableIT_FLT2

LL_HRTIM_DisableIT_FLT2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 2 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT2IE LL_HRTIM_DisableIT_FLT2

LL_HRTIM_IsEnabledIT_FLT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 2 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT2IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT2IE LL_HRTIM_IsEnabledIT_FLT2

LL_HRTIM_EnableIT_FLT3

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 3 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT3IE LL_HRTIM_EnableIT_FLT3

LL_HRTIM_DisableIT_FLT3

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 3 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT3IE LL_HRTIM_DisableIT_FLT3

LL_HRTIM_IsEnabledIT_FLT3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 3 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT3IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT3IE LL_HRTIM_IsEnabledIT_FLT3

LL_HRTIM_EnableIT_FLT4

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 4 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT4IE LL_HRTIM_EnableIT_FLT4

LL_HRTIM_DisableIT_FLT4

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 4 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT4IE LL_HRTIM_DisableIT_FLT4

LL_HRTIM_IsEnabledIT_FLT4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 4 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT4IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT4IE LL_HRTIM_IsEnabledIT_FLT4

LL_HRTIM_EnableIT_FLT5

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 5 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT5IE LL_HRTIM_EnableIT_FLT5

LL_HRTIM_DisableIT_FLT5

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 5 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT5IE LL_HRTIM_DisableIT_FLT5

LL_HRTIM_IsEnabledIT_FLT5

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 5 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT5IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT5IE LL_HRTIM_IsEnabledIT_FLT5

LL_HRTIM_EnableIT_FLT6

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT6 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 6 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT6IE LL_HRTIM_EnableIT_FLT6

LL_HRTIM_DisableIT_FLT6

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT6 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 6 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT6IE LL_HRTIM_DisableIT_FLT6

LL_HRTIM_IsEnabledIT_FLT6

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT6 (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 6 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT6IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT6IE LL_HRTIM_IsEnabledIT_FLT6

LL_HRTIM_EnableIT_SYSFLT

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Enable the system fault interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SYSFLTIE LL_HRTIM_EnableIT_SYSFLT

LL_HRTIM_DisableIT_SYSFLT

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Disable the system fault interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SYSFLTIE LL_HRTIM_DisableIT_SYSFLT

LL_HRTIM_IsEnabledIT_SYSFLT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the system fault interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYSFLTIE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER SYSFLTIE LL_HRTIM_IsEnabledIT_SYSFLT

LL_HRTIM_EnableIT_DLLRDY

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_DLLRDY (HRTIM_TypeDef * HRTIMx)

Function description

Enable the DLL ready interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DLLRDYIE LL_HRTIM_EnableIT_DLLRDY

LL_HRTIM_DisableIT_DLLRDY

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_DLLRDY (HRTIM_TypeDef * HRTIMx)

Function description

Disable the DLL ready interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DLLRDYIE LL_HRTIM_DisableIT_DLLRDY

LL_HRTIM_IsEnabledIT_DLLRDY

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_DLLRDY (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the DLL ready interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of DLLRDYIE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER DLLRDYIE LL_HRTIM_IsEnabledIT_DLLRDY

LL_HRTIM_EnableIT_BMPER

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Enable the burst mode period interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BMPERIE LL_HRTIM_EnableIT_BMPER

LL_HRTIM_DisableIT_BMPER

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Disable the burst mode period interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BMPERIE LL_HRTIM_DisableIT_BMPER

LL_HRTIM_IsEnabledIT_BMPER

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the burst mode period interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BMPERIE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER BMPERIE LL_HRTIM_IsEnabledIT_BMPER

LL_HRTIM_EnableIT_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Enable the synchronization input interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCIE LL_HRTIM_EnableIT_SYNC

LL_HRTIM_DisableIT_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Disable the synchronization input interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCIE LL_HRTIM_DisableIT_SYNC

LL_HRTIM_IsEnabledIT_SYNC

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the synchronization input interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYNCIE bit in HRTIM_MDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER SYNCIE LL_HRTIM_IsEnabledIT_SYNC

LL_HRTIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the update interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDIE LL_HRTIM_EnableIT_UPDATE
- TIMxDIER UPDIE LL_HRTIM_EnableIT_UPDATE

LL_HRTIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the update interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDIE LL_HRTIM_DisableIT_UPDATE
- TIMxDIER UPDIE LL_HRTIM_DisableIT_UPDATE

LL_HRTIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the update interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MUPDIE/UPDIE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MUPDIE LL_HRTIM_IsEnabledIT_UPDATE
- TIMxDIER UPDIE LL_HRTIM_IsEnabledIT_UPDATE

LL_HRTIM_EnableIT_REP

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the repetition interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPIE LL_HRTIM_EnableIT_REP
- TIMxDIER REPIE LL_HRTIM_EnableIT_REP

LL_HRTIM_DisableIT_REP

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the repetition interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPIE LL_HRTIM_DisableIT_REP
- TIMxDIER REPIE LL_HRTIM_DisableIT_REP

LL_HRTIM_IsEnabledIT_REP

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the repetition interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MREPIE/REPIE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MREPIE LL_HRTIM_IsEnabledIT_REP
- TIMxDIER REPIE LL_HRTIM_IsEnabledIT_REP

LL_HRTIM_EnableIT_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1IE LL_HRTIM_EnableIT_CMP1
- TIMxDIER CMP1IE LL_HRTIM_EnableIT_CMP1

LL_HRTIM_DisableIT_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1IE LL_HRTIM_DisableIT_CMP1
- TIMxDIER CMP1IE LL_HRTIM_DisableIT_CMP1

LL_HRTIM_IsEnabledIT_CMP1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 1 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP1IE/CMP1IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP1IE LL_HRTIM_IsEnabledIT_CMP1
- TIMxDIER CMP1IE LL_HRTIM_IsEnabledIT_CMP1

LL_HRTIM_EnableIT_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2IE LL_HRTIM_EnableIT_CMP2
- TIMxDIER CMP2IE LL_HRTIM_EnableIT_CMP2

LL_HRTIM_DisableIT_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2IE LL_HRTIM_DisableIT_CMP2
- TIMxDIER CMP2IE LL_HRTIM_DisableIT_CMP2

LL_HRTIM_IsEnabledIT_CMP2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 2 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP2IE/CMP2IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP2IE LL_HRTIM_IsEnabledIT_CMP2
- TIMxDIER CMP2IE LL_HRTIM_IsEnabledIT_CMP2

LL_HRTIM_EnableIT_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 3 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3IE LL_HRTIM_EnableIT_CMP3
- TIMxDIER CMP3IE LL_HRTIM_EnableIT_CMP3

LL_HRTIM_DisableIT_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 3 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3IE LL_HRTIM_DisableIT_CMP3
- TIMxDIER CMP3IE LL_HRTIM_DisableIT_CMP3

LL_HRTIM_IsEnabledIT_CMP3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 3 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP3IE/CMP3IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP3IE LL_HRTIM_IsEnabledIT_CMP3
- TIMxDIER CMP3IE LL_HRTIM_IsEnabledIT_CMP3

LL_HRTIM_EnableIT_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 4 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4IE LL_HRTIM_EnableIT_CMP4
- TIMxDIER CMP4IE LL_HRTIM_EnableIT_CMP4

LL_HRTIM_DisableIT_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 4 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4IE LL_HRTIM_DisableIT_CMP4
- TIMxDIER CMP4IE LL_HRTIM_DisableIT_CMP4

LL_HRTIM_IsEnabledIT_CMP4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 4 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP4IE/CMP4IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP4IE LL_HRTIM_IsEnabledIT_CMP4
- TIMxDIER CMP4IE LL_HRTIM_IsEnabledIT_CMP4

LL_HRTIM_EnableIT_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1IE LL_HRTIM_EnableIT_CPT1

LL_HRTIM_DisableIT_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1IE LL_HRTIM_DisableIT_CPT1

LL_HRTIM_IsEnabledIT_CPT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 1 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of CPT1IE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT1IE LL_HRTIM_IsEnabledIT_CPT1

LL_HRTIM_EnableIT_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2IE LL_HRTIM_EnableIT_CPT2

LL_HRTIM_DisableIT_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2IE LL_HRTIM_DisableIT_CPT2

LL_HRTIM_IsEnabledIT_CPT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 2 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of CPT2IE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT2IE LL_HRTIM_IsEnabledIT_CPT2

LL_HRTIM_EnableIT_SET1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1IE LL_HRTIM_EnableIT_SET1

LL_HRTIM_DisableIT_SET1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1IE LL_HRTIM_DisableIT_SET1

LL_HRTIM_IsEnabledIT_SET1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 set interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SET1xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET1IE LL_HRTIM_IsEnabledIT_SET1

LL_HRTIM_EnableIT_RST1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1IE LL_HRTIM_EnableIT_RST1

LL_HRTIM_DisableIT_RST1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1IE LL_HRTIM_DisableIT_RST1

LL_HRTIM_IsEnabledIT_RST1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 reset interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RST1xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST1IE LL_HRTIM_IsEnabledIT_RST1

LL_HRTIM_EnableIT_SET2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2IE LL_HRTIM_EnableIT_SET2

LL_HRTIM_DisableIT_SET2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2IE LL_HRTIM_DisableIT_SET2

LL_HRTIM_IsEnabledIT_SET2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 set interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SET2xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET2IE LL_HRTIM_IsEnabledIT_SET2

LL_HRTIM_EnableIT_RST2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2IE LL_HRTIM_EnableIT_RST2

LL_HRTIM_DisableIT_RST2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2IE LL_HRTIM_DisableIT_RST2

LL_HRTIM_IsEnabledIT_RST2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 reset LL_HRTIM_IsEnabledIT_RST2 is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RST2xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST2IE LL_HRTIM_DisableIT_RST2

LL_HRTIM_EnableIT_RST

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the reset/roll-over interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTIE LL_HRTIM_EnableIT_RST

LL_HRTIM_DisableIT_RST

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the reset/roll-over interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTIE LL_HRTIM_DisableIT_RST

LL_HRTIM_IsEnabledIT_RST

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the reset/roll-over interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RSTIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RSTIE LL_HRTIM_IsEnabledIT_RST

LL_HRTIM_EnableIT_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the delayed protection interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTIE LL_HRTIM_EnableIT_DLYPRT

LL_HRTIM_DisableIT_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the delayed protection interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTIE LL_HRTIM_DisableIT_DLYPRT

LL_HRTIM_IsEnabledIT_DLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the delayed protection interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DLYPRTIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTIE LL_HRTIM_IsEnabledIT_DLYPRT

LL_HRTIM_EnableDMAReq_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Enable the synchronization input DMA request.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCDE LL_HRTIM_EnableDMAReq_SYNC

LL_HRTIM_DisableDMAReq_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Disable the synchronization input DMA request.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCDE LL_HRTIM_DisableDMAReq_SYNC

LL_HRTIM_IsEnabledDMAReq_SYNC

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the synchronization input DMA request is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYNCDE bit in HRTIM_MDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER SYNCDE LL_HRTIM_IsEnabledDMAReq_SYNC

LL_HRTIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the update DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDDE LL_HRTIM_EnableDMAReq_UPDATE
- TIMxDIER UPDDE LL_HRTIM_EnableDMAReq_UPDATE

LL_HRTIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the update DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDDE LL_HRTIM_DisableDMAReq_UPDATE
- TIMxDIER UPDDE LL_HRTIM_DisableDMAReq_UPDATE

LL_HRTIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the update DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MUPDDE/UPDDE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MUPDDE LL_HRTIM_IsEnabledDMAReq_UPDATE
- TIMxDIER UPDDE LL_HRTIM_IsEnabledDMAReq_UPDATE

LL_HRTIM_EnableDMAReq_REP

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the repetition DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPDE LL_HRTIM_EnableDMAReq_REP
- TIMxDIER REPDE LL_HRTIM_EnableDMAReq_REP

LL_HRTIM_DisableDMAReq_REP

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the repetition DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPDE LL_HRTIM_DisableDMAReq_REP
- TIMxDIER REPDE LL_HRTIM_DisableDMAReq_REP

LL_HRTIM_IsEnabledDMAReq_REP

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the repetition DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MREPDE/REPDE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MREPDE LL_HRTIM_IsEnabledDMAReq_REP
- TIMxDIER REPDE LL_HRTIM_IsEnabledDMAReq_REP

LL_HRTIM_EnableDMAReq_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1DE LL_HRTIM_EnableDMAReq_CMP1
- TIMxDIER CMP1DE LL_HRTIM_EnableDMAReq_CMP1

LL_HRTIM_DisableDMAReq_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1DE LL_HRTIM_DisableDMAReq_CMP1
- TIMxDIER CMP1DE LL_HRTIM_DisableDMAReq_CMP1

LL_HRTIM_IsEnabledDMAReq_CMP1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 1 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP1DE/CMP1DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP1DE LL_HRTIM_IsEnabledDMAReq_CMP1
- TIMxDIER CMP1DE LL_HRTIM_IsEnabledDMAReq_CMP1

LL_HRTIM_EnableDMAReq_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2DE LL_HRTIM_EnableDMAReq_CMP2
- TIMxDIER CMP2DE LL_HRTIM_EnableDMAReq_CMP2

LL_HRTIM_DisableDMAReq_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2DE LL_HRTIM_DisableDMAReq_CMP2
- TIMxDIER CMP2DE LL_HRTIM_DisableDMAReq_CMP2

LL_HRTIM_IsEnabledDMAReq_CMP2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 2 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP2DE/CMP2DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP2DE LL_HRTIM_IsEnabledDMAReq_CMP2
- TIMxDIER CMP2DE LL_HRTIM_IsEnabledDMAReq_CMP2

LL_HRTIM_EnableDMAReq_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 3 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3DE LL_HRTIM_EnableDMAReq_CMP3
- TIMxDIER CMP3DE LL_HRTIM_EnableDMAReq_CMP3

LL_HRTIM_DisableDMAReq_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 3 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3DE LL_HRTIM_DisableDMAReq_CMP3
- TIMxDIER CMP3DE LL_HRTIM_DisableDMAReq_CMP3

LL_HRTIM_IsEnabledDMAReq_CMP3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 3 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP3DE/CMP3DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP3DE LL_HRTIM_IsEnabledDMAReq_CMP3
- TIMxDIER CMP3DE LL_HRTIM_IsEnabledDMAReq_CMP3

LL_HRTIM_EnableDMAReq_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 4 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4DE LL_HRTIM_EnableDMAReq_CMP4
- TIMxDIER CMP4DE LL_HRTIM_EnableDMAReq_CMP4

LL_HRTIM_DisableDMAReq_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 4 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4DE LL_HRTIM_DisableDMAReq_CMP4
- TIMxDIER CMP4DE LL_HRTIM_DisableDMAReq_CMP4

LL_HRTIM_IsEnabledDMAReq_CMP4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 4 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of MCMP4DE/CMP4DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP4DE LL_HRTIM_IsEnabledDMAReq_CMP4
- TIMxDIER CMP4DE LL_HRTIM_IsEnabledDMAReq_CMP4

LL_HRTIM_EnableDMAReq_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1DE LL_HRTIM_EnableDMAReq_CPT1

LL_HRTIM_DisableDMAReq_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the capture 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1DE LL_HRTIM_DisableDMAReq_CPT1

LL_HRTIM_IsEnabledDMAReq_CPT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 1 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of CPT1DE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT1DE LL_HRTIM_IsEnabledDMAReq_CPT1

LL_HRTIM_EnableDMAReq_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2DE LL_HRTIM_EnableDMAReq_CPT2

LL_HRTIM_DisableDMAReq_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the capture 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2DE LL_HRTIM_DisableDMAReq_CPT2

LL_HRTIM_IsEnabledDMAReq_CPT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 2 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of CPT2DE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT2DE LL_HRTIM_IsEnabledDMAReq_CPT2

LL_HRTIM_EnableDMAReq_SET1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1DE LL_HRTIM_EnableDMAReq_SET1

LL_HRTIM_DisableDMAReq_SET1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1DE LL_HRTIM_DisableDMAReq_SET1

LL_HRTIM_IsEnabledDMAReq_SET1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 set DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SET1xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET1DE LL_HRTIM_IsEnabledDMAReq_SET1

LL_HRTIM_EnableDMAReq_RST1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1DE LL_HRTIM_EnableDMAReq_RST1

LL_HRTIM_DisableDMAReq_RST1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1DE LL_HRTIM_DisableDMAReq_RST1

LL_HRTIM_IsEnabledDMAReq_RST1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 reset interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RST1xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST1DE LL_HRTIM_IsEnabledDMAReq_RST1

LL_HRTIM_EnableDMAReq_SET2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2DE LL_HRTIM_EnableDMAReq_SET2

LL_HRTIM_DisableDMAReq_SET2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2DE LL_HRTIM_DisableDMAReq_SET2

LL_HRTIM_IsEnabledDMAReq_SET2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 set DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of SET2xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET2DE LL_HRTIM_IsEnabledDMAReq_SET2

LL_HRTIM_EnableDMAReq_RST2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2DE LL_HRTIM_EnableDMAReq_RST2

LL_HRTIM_DisableDMAReq_RST2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2DE LL_HRTIM_DisableDMAReq_RST2

LL_HRTIM_IsEnabledDMAReq_RST2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 reset DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RST2xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST2DE LL_HRTIM_IsEnabledDMAReq_RST2

LL_HRTIM_EnableDMAReq_RST

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the reset/roll-over DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTDE LL_HRTIM_EnableDMAReq_RST

LL_HRTIM_DisableDMAReq_RST

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the reset/roll-over DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTDE LL_HRTIM_DisableDMAReq_RST

LL_HRTIM_IsEnabledDMAReq_RST

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the reset/roll-over DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of RSTDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RSTDE LL_HRTIM_IsEnabledDMAReq_RST

LL_HRTIM_EnableDMAReq_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the delayed protection DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTDE LL_HRTIM_EnableDMAReq_DLYPRT

LL_HRTIM_DisableDMAReq_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the delayed protection DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTDE LL_HRTIM_DisableDMAReq_DLYPRT

LL_HRTIM_IsEnabledDMAReq_DLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the delayed protection DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
  - LL_HRTIM_TIMER_F

Return values

- **State:** of DLYPRTDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTDE LL_HRTIM_IsEnabledDMAReq_DLYPRT

LL_HRTIM_DeInit

Function name

ErrorStatus LL_HRTIM_DeInit (HRTIM_TypeDef * HRTIMx)

Function description

Set HRTIM instance registers to their reset values.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **ErrorStatus:** enumeration value:
  - SUCCESS: HRTIMx registers are de-initialized
  - ERROR: invalid HRTIMx instance

78.2 HRTIM Firmware driver defines

The following section lists the various define and macros of the module.

78.2.1 HRTIM

HRTIM

Greater than compare PWM Mode

LL_HRTIM_GTCMP1_EQUAL

event is generated when counter is equal to compare value

LL_HRTIM_GTCMP1_GREATER

event is generated when counter is greater than compare value

LL_HRTIM_GTCMP3_EQUAL

event is generated when counter is equal to compare value

LL_HRTIM_GTCMP3_GREATER

event is generated when counter is greater than compare value

Enabling the Dual Channel DAC Triggering

LL_HRTIM_DCDE_DISABLED

Dual Channel DAC trigger is generated on counter reset or roll-over event

LL_HRTIM_DCDE_ENABLED

Dual Channel DAC trigger is generated on output 1 set event

Dual Channel DAC Reset Trigger

LL_HRTIM_DCDR_COUNTER

Dual Channel DAC trigger is generated on counter reset or roll-over event

LL_HRTIM_DCDR_OUT1SET

Dual Channel DAC trigger is generated on output 1 set event

Dual Channel DAC Step trigger

LL_HRTIM_DCDS_CMP2

trigger is generated on compare 2 event

LL_HRTIM_DCDS_OUT1RST

trigger is generated on output 1 reset event

Counter Mode

LL_HRTIM_COUNTING_MODE_UP

counter is operating in up-counting mode

LL_HRTIM_COUNTING_MODE_UP_DOWN

counter is operating in up-down counting mode

counter Mode

LL_HRTIM_ROLLOVER_MODE_PER

Event generated when counter reaches period value ('crest' mode)

LL_HRTIM_ROLLOVER_MODE_RST

Event generated when counter equals 0 ('valley' mode)

LL_HRTIM_ROLLOVER_MODE_BOTH

Event generated when counter reach both conditions (0 or HRTIM_PERxR value)

ADC TRIGGER

LL_HRTIM_ADCTRIG_1

ADC trigger 1 identifier

LL_HRTIM_ADCTRIG_2

ADC trigger 2 identifier

LL_HRTIM_ADCTRIG_3

ADC trigger 3 identifier

LL_HRTIM_ADCTRIG_4

ADC trigger 4 identifier

LL_HRTIM_ADCTRIG_5

ADC trigger 5 identifier

LL_HRTIM_ADCTRIG_6

ADC trigger 6 identifier

LL_HRTIM_ADCTRIG_7

ADC trigger 7 identifier

LL_HRTIM_ADCTRIG_8

ADC trigger 8 identifier

LL_HRTIM_ADCTRIG_9

ADC trigger 9 identifier

LL_HRTIM_ADCTRIG_10

ADC trigger 10 identifier

ADC TRIGGER 1/3 SOURCE

LL_HRTIM_ADCTRIG_SRC13_NONE

No ADC trigger event

LL_HRTIM_ADCTRIG_SRC13_MCMP1

ADC Trigger on master compare 1

LL_HRTIM_ADCTRIG_SRC13_MCMP2

ADC Trigger on master compare 2

LL_HRTIM_ADCTRIG_SRC13_MCMP3

ADC Trigger on master compare 3

LL_HRTIM_ADCTRIG_SRC13_MCMP4

ADC Trigger on master compare 4

LL_HRTIM_ADCTRIG_SRC13_MPER

ADC Trigger on master period

LL_HRTIM_ADCTRIG_SRC13_EEV1

ADC Trigger on external event 1

LL_HRTIM_ADCTRIG_SRC13_EEV2

ADC Trigger on external event 2

LL_HRTIM_ADCTRIG_SRC13_EEV3

ADC Trigger on external event 3

LL_HRTIM_ADCTRIG_SRC13_EEV4

ADC Trigger on external event 4

LL_HRTIM_ADCTRIG_SRC13_EEV5

ADC Trigger on external event 5

LL_HRTIM_ADCTRIG_SRC13_TIMFCMP2

ADC Trigger on Timer F compare 2

LL_HRTIM_ADCTRIG_SRC13_TIMACMP3

ADC Trigger on Timer A compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMACMP4

ADC Trigger on Timer A compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMAPER

ADC Trigger on Timer A period

LL_HRTIM_ADCTRIG_SRC13_TIMARST

ADC Trigger on Timer A reset

LL_HRTIM_ADCTRIG_SRC13_TIMFCMP3

ADC Trigger on Timer F compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMBCMP3

ADC Trigger on Timer B compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMBCMP4

ADC Trigger on Timer B compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMBPER

ADC Trigger on Timer B period

LL_HRTIM_ADCTRIG_SRC13_TIMBRST

ADC Trigger on Timer B reset

LL_HRTIM_ADCTRIG_SRC13_TIMFCMP4

ADC Trigger on Timer F compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMCCMP3

ADC Trigger on Timer C compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMCCMP4

ADC Trigger on Timer C compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMCPER

ADC Trigger on Timer C period

LL_HRTIM_ADCTRIG_SRC13_TIMFPER

ADC Trigger on Timer F period

LL_HRTIM_ADCTRIG_SRC13_TIMDCMP3

ADC Trigger on Timer D compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMDCMP4

ADC Trigger on Timer D compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMDPER

ADC Trigger on Timer D period

LL_HRTIM_ADCTRIG_SRC13_TIMFRST

ADC Trigger on Timer F reset

LL_HRTIM_ADCTRIG_SRC13_TIMECMP3

ADC Trigger on Timer E compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMECMP4

ADC Trigger on Timer E compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMEPER

ADC Trigger on Timer E period

ADC TRIGGER 2/4 SOURCE

LL_HRTIM_ADCTRIG_SRC24_NONE

No ADC trigger event

LL_HRTIM_ADCTRIG_SRC24_MCMP1

ADC Trigger on master compare 1

LL_HRTIM_ADCTRIG_SRC24_MCMP2

ADC Trigger on master compare 2

LL_HRTIM_ADCTRIG_SRC24_MCMP3

ADC Trigger on master compare 3

LL_HRTIM_ADCTRIG_SRC24_MCMP4

ADC Trigger on master compare 4

LL_HRTIM_ADCTRIG_SRC24_MPER

ADC Trigger on master period

LL_HRTIM_ADCTRIG_SRC24_EEV6

ADC Trigger on external event 6

LL_HRTIM_ADCTRIG_SRC24_EEV7

ADC Trigger on external event 7

LL_HRTIM_ADCTRIG_SRC24_EEV8

ADC Trigger on external event 8

LL_HRTIM_ADCTRIG_SRC24_EEV9

ADC Trigger on external event 9

LL_HRTIM_ADCTRIG_SRC24_EEV10

ADC Trigger on external event 10

LL_HRTIM_ADCTRIG_SRC24_TIMACMP2

ADC Trigger on Timer A compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMFCMP2

ADC Trigger on Timer F compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMACMP4

ADC Trigger on Timer A compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMAPER

ADC Trigger on Timer A period

LL_HRTIM_ADCTRIG_SRC24_TIMBCMP2

ADC Trigger on Timer B compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMFCMP3

ADC Trigger on Timer F compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMBCMP4

ADC Trigger on Timer B compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMBPER

ADC Trigger on Timer B period

LL_HRTIM_ADCTRIG_SRC24_TIMCCMP2

ADC Trigger on Timer C compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMFCMP4

ADC Trigger on Timer F compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMCCMP4

ADC Trigger on Timer C compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMCPER

ADC Trigger on Timer C period

LL_HRTIM_ADCTRIG_SRC24_TIMCRST

ADC Trigger on Timer C reset

LL_HRTIM_ADCTRIG_SRC24_TIMDCMP2

ADC Trigger on Timer D compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMFPER

ADC Trigger on Timer F period

LL_HRTIM_ADCTRIG_SRC24_TIMDCMP4

ADC Trigger on Timer D compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMDPER

ADC Trigger on Timer D period

LL_HRTIM_ADCTRIG_SRC24_TIMDRST

ADC Trigger on Timer D reset

LL_HRTIM_ADCTRIG_SRC24_TIMECMP2

ADC Trigger on Timer E compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMECMP3

ADC Trigger on Timer E compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMECMP4

ADC Trigger on Timer E compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMERST

ADC Trigger on Timer E reset

ADC TRIGGER UPDATE

LL_HRTIM_ADCTRIG_UPDATE_MASTER

HRTIM_ADCxR register update is triggered by the Master timer

LL_HRTIM_ADCTRIG_UPDATE_TIMER_A

HRTIM_ADCxR register update is triggered by the Timer A

LL_HRTIM_ADCTRIG_UPDATE_TIMER_B

HRTIM_ADCxR register update is triggered by the Timer B

LL_HRTIM_ADCTRIG_UPDATE_TIMER_C

HRTIM_ADCxR register update is triggered by the Timer C

LL_HRTIM_ADCTRIG_UPDATE_TIMER_D

HRTIM_ADCxR register update is triggered by the Timer D

LL_HRTIM_ADCTRIG_UPDATE_TIMER_E

HRTIM_ADCxR register update is triggered by the Timer E

LL_HRTIM_ADCTRIG_UPDATE_TIMER_F

HRTIM_ADCxR register update is triggered by the Timer F

BURST MODE CLOCK SOURCE

LL_HRTIM_BM_CLKSRC_MASTER

Master timer counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_A

Timer A counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_B

Timer B counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_C

Timer C counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_D

Timer D counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_E

Timer E counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_F

Timer F counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIM16_OC

On-chip Event 1 (BMClk[1]), acting as a burst mode counter clock

LL_HRTIM_BM_CLKSRC_TIM17_OC

On-chip Event 2 (BMClk[2]), acting as a burst mode counter clock

LL_HRTIM_BM_CLKSRC_TIM7_TRGO

On-chip Event 3 (BMClk[3]), acting as a burst mode counter clock

LL_HRTIM_BM_CLKSRC_FHRTIM

Prescaled fHRTIM clock is used as clock source for the burst mode counter

BURST MODE OPERATING MODE

LL_HRTIM_BM_MODE_SINGLESHOT

Burst mode operates in single shot mode

LL_HRTIM_BM_MODE_CONTINOUS

Burst mode operates in continuous mode

BURST MODE PRESCALER

LL_HRTIM_BM_PRESCALER_DIV1

fBRST = fHRTIM

LL_HRTIM_BM_PRESCALER_DIV2

fBRST = fHRTIM/2

LL_HRTIM_BM_PRESCALER_DIV4

fBRST = fHRTIM/4

LL_HRTIM_BM_PRESCALER_DIV8

fBRST = fHRTIM/8

LL_HRTIM_BM_PRESCALER_DIV16

fBRST = fHRTIM/16

LL_HRTIM_BM_PRESCALER_DIV32

fBRST = fHRTIM/32

LL_HRTIM_BM_PRESCALER_DIV64

fBRST = fHRTIM/64

LL_HRTIM_BM_PRESCALER_DIV128

fBRST = fHRTIM/128

LL_HRTIM_BM_PRESCALER_DIV256

fBRST = fHRTIM/256

LL_HRTIM_BM_PRESCALER_DIV512

fBRST = fHRTIM/512

LL_HRTIM_BM_PRESCALER_DIV1024

fBRST = fHRTIM/1024

LL_HRTIM_BM_PRESCALER_DIV2048

fBRST = fHRTIM/2048

LL_HRTIM_BM_PRESCALER_DIV4096

fBRST = fHRTIM/4096

LL_HRTIM_BM_PRESCALER_DIV8192

fBRST = fHRTIM/8192

LL_HRTIM_BM_PRESCALER_DIV16384

fBRST = fHRTIM/16384

LL_HRTIM_BM_PRESCALER_DIV32768

fBRST = fHRTIM/32768

HRTIM BURST MODE STATUS

LL_HRTIM_BM_STATUS_NORMAL

Normal operation

LL_HRTIM_BM_STATUS_BURST_ONGOING

Burst operation on-going

HRTIM BURST MODE TRIGGER

LL_HRTIM_BM_TRIG_NONE

No trigger

LL_HRTIM_BM_TRIG_MASTER_RESET

Master timer reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_REPETITION

Master timer repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP1

Master timer compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP2

Master timer compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP3

Master timer compare 3 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP4

Master timer compare 4 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_RESET

Timer A reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_REPETITION

Timer A repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_CMP1

Timer A compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_CMP2

Timer A compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_RESET

Timer B reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_REPETITION

Timer B repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_CMP1

Timer B compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_CMP2

Timer B compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_RESET

Timer C resetevent is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_REPETITION

Timer C repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_CMP1

Timer C compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_CMP2

Timer C compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_RESET

Timer D reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_REPETITION

Timer D repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_CMP1

Timer D compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_CMP2

Timer D compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_RESET

Timer E reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_REPETITION

Timer E repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_CMP1

Timer E compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_CMP2

Timer E compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMF_RESET

Timer F reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMF_REPETITION

Timer F repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMF_CMP1

Timer F compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMF_CMP2

Timer F compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_EVENT7

Timer A period following an external event 7 (conditioned by TIMA filters) is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_EVENT8

Timer D period following an external event 8 (conditioned by TIMD filters) is starting the burst mode operation

LL_HRTIM_BM_TRIG_EVENT_7

External event 7 conditioned by TIMA filters is starting the burst mode operation

LL_HRTIM_BM_TRIG_EVENT_8

External event 8 conditioned by TIMD filters is starting the burst mode operation

LL_HRTIM_BM_TRIG_EVENT_ONCHIP

A rising edge on an on-chip Event (for instance from GP timer or comparator) triggers the burst mode operation

BURST DMA

LL_HRTIM_BURSTDMA_NONE

No register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCR

MCR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MICR

MICR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MDIER

MDIER register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCNT

MCNTR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MPER

MPER register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MREP

MREPR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP1

MCMP1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP2

MCMP2R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP3

MCMP3R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP4

MCMP4R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMMCR

TIMxCR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMICR

TIMxICR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMDIER

TIMxDIER register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCNT

CNTxCR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMPER

PERxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMREP

REPxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP1

CMP1xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP2

CMP2xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP3

CMP3xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP4

CMP4xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMDTR

DTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMSET1R

SET1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMRST1R

RST1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMSET2R

SET2R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMRST2R

RST1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMEEFR1

EEFxR1 register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMEEFR2

EEFxR2 register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMRSTR

RSTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCHPR

CHPxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMOUTR

OUTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMFLTR

FLTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_CR2

TIMxCR2 register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_EEFR3

EEFxR3 register is updated by Burst DMA accesses

BURST MODE

LL_HRTIM_BURSTMODE_MAINTAINCLOCK

Timer counter clock is maintained and the timer operates normally

LL_HRTIM_BURSTMODE_RESETCOUNTER

Timer counter clock is stopped and the counter is reset

DLL CALIBRATION RATE

LL_HRTIM_DLLCALIBRATION_RATE_0

Periodic DLL calibration: T = 1048576U * tHRTIM (6.168 ms)

LL_HRTIM_DLLCALIBRATION_RATE_1

Periodic DLL calibration: T = 131072U * tHRTIM (0.771 ms)

LL_HRTIM_DLLCALIBRATION_RATE_2

Periodic DLL calibration: T = 16384U * tHRTIM (0.096 ms)

LL_HRTIM_DLLCALIBRATION_RATE_3

Periodic DLL calibration: T = 2048U * tHRTIM (0.012 ms)

CAPTURE TRIGGER

LL_HRTIM_CAPTURETRIG_NONE

Capture trigger is disabled

LL_HRTIM_CAPTURETRIG_SW

The sw event triggers the Capture

LL_HRTIM_CAPTURETRIG_UPDATE

The update event triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_1

The External event 1 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_2

The External event 2 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_3

The External event 3 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_4

The External event 4 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_5

The External event 5 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_6

The External event 6 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_7

The External event 7 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_8

The External event 8 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_9

The External event 9 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_10

The External event 10 triggers the Capture

LL_HRTIM_CAPTURETRIG_TA1_SET

Capture is triggered by TA1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TA1_RESET

Capture is triggered by TA1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMA_CMP1

Timer A Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMA_CMP2

Timer A Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TB1_SET

Capture is triggered by TB1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TB1_RESET

Capture is triggered by TB1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMB_CMP1

Timer B Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMB_CMP2

Timer B Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TC1_SET

Capture is triggered by TC1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TC1_RESET

Capture is triggered by TC1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMC_CMP1

Timer C Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMC_CMP2

Timer C Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TD1_SET

Capture is triggered by TD1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TD1_RESET

Capture is triggered by TD1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMD_CMP1

Timer D Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMD_CMP2

Timer D Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TE1_SET

Capture is triggered by TE1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TE1_RESET

Capture is triggered by TE1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIME_CMP1

Timer E Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIME_CMP2

Timer E Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TF1_SET

Capture is triggered by TF1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TF1_RESET

Capture is triggered by TF1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMF_CMP1

Timer F Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMF_CMP2

Timer F Compare 2 triggers Capture

CAPTURE UNIT ID

LL_HRTIM_CAPTUREUNIT_1

Capture unit 1 identifier

LL_HRTIM_CAPTUREUNIT_2

Capture unit 2 identifier

CHOPPER MODE DUTY CYCLE

LL_HRTIM_CHP_DUTYCYCLE_0

Only 1st pulse is present

LL_HRTIM_CHP_DUTYCYCLE_125

Duty cycle of the carrier signal is 12.5 %

LL_HRTIM_CHP_DUTYCYCLE_250

Duty cycle of the carrier signal is 25 %

LL_HRTIM_CHP_DUTYCYCLE_375

Duty cycle of the carrier signal is 37.5 %

LL_HRTIM_CHP_DUTYCYCLE_500

Duty cycle of the carrier signal is 50 %

LL_HRTIM_CHP_DUTYCYCLE_625

Duty cycle of the carrier signal is 62.5 %

LL_HRTIM_CHP_DUTYCYCLE_750

Duty cycle of the carrier signal is 75 %

LL_HRTIM_CHP_DUTYCYCLE_875

Duty cycle of the carrier signal is 87.5 %

CHOPPER MODE PRESCALER

LL_HRTIM_CHP_PRESCALER_DIV16

fCHPFRQ = fHRTIM / 16

LL_HRTIM_CHP_PRESCALER_DIV32

fCHPFRQ = fHRTIM / 32

LL_HRTIM_CHP_PRESCALER_DIV48

fCHPFRQ = fHRTIM / 48

LL_HRTIM_CHP_PRESCALER_DIV64

fCHPFRQ = fHRTIM / 64

LL_HRTIM_CHP_PRESCALER_DIV80

fCHPFRQ = fHRTIM / 80

LL_HRTIM_CHP_PRESCALER_DIV96

fCHPFRQ = fHRTIM / 96

LL_HRTIM_CHP_PRESCALER_DIV112

fCHPFRQ = fHRTIM / 112

LL_HRTIM_CHP_PRESCALER_DIV128

fCHPFRQ = fHRTIM / 128

LL_HRTIM_CHP_PRESCALER_DIV144

fCHPFRQ = fHRTIM / 144

LL_HRTIM_CHP_PRESCALER_DIV160

fCHPFRQ = fHRTIM / 160

LL_HRTIM_CHP_PRESCALER_DIV176

fCHPFRQ = fHRTIM / 176

LL_HRTIM_CHP_PRESCALER_DIV192

fCHPFRQ = fHRTIM / 192

LL_HRTIM_CHP_PRESCALER_DIV208

fCHPFRQ = fHRTIM / 208

LL_HRTIM_CHP_PRESCALER_DIV224

fCHPFRQ = fHRTIM / 224

LL_HRTIM_CHP_PRESCALER_DIV240

fCHPFRQ = fHRTIM / 240

LL_HRTIM_CHP_PRESCALER_DIV256

fCHPFRQ = fHRTIM / 256

CHOPPER MODE PULSE WIDTH

LL_HRTIM_CHP_PULSEWIDTH_16

tSTPW = tHRTIM x 16

LL_HRTIM_CHP_PULSEWIDTH_32

tSTPW = tHRTIM x 32

LL_HRTIM_CHP_PULSEWIDTH_48

tSTPW = tHRTIM x 48

LL_HRTIM_CHP_PULSEWIDTH_64

tSTPW = tHRTIM x 64

LL_HRTIM_CHP_PULSEWIDTH_80

tSTPW = tHRTIM x 80

LL_HRTIM_CHP_PULSEWIDTH_96

tSTPW = tHRTIM x 96

LL_HRTIM_CHP_PULSEWIDTH_112

tSTPW = tHRTIM x 112

LL_HRTIM_CHP_PULSEWIDTH_128

tSTPW = tHRTIM x 128

LL_HRTIM_CHP_PULSEWIDTH_144

tSTPW = tHRTIM x 144

LL_HRTIM_CHP_PULSEWIDTH_160

tSTPW = tHRTIM x 160

LL_HRTIM_CHP_PULSEWIDTH_176

tSTPW = tHRTIM x 176

LL_HRTIM_CHP_PULSEWIDTH_192

tSTPW = tHRTIM x 192

LL_HRTIM_CHP_PULSEWIDTH_208

tSTPW = tHRTIM x 208

LL_HRTIM_CHP_PULSEWIDTH_224

tSTPW = tHRTIM x 224

LL_HRTIM_CHP_PULSEWIDTH_240

tSTPW = tHRTIM x 240

LL_HRTIM_CHP_PULSEWIDTH_256

tSTPW = tHRTIM x 256

COMPARE MODE

LL_HRTIM_COMPAREMODE_REGULAR

standard compare mode

LL_HRTIM_COMPAREMODE_DELAY_NOTIMEOUT

Compare event generated only if a capture has occurred

LL_HRTIM_COMPAREMODE_DELAY_CMP1

Compare event generated if a capture has occurred or after a Compare 1 match (timeout if capture event is missing)

LL_HRTIM_COMPAREMODE_DELAY_CMP3

Compare event generated if a capture has occurred or after a Compare 3 match (timeout if capture event is missing)

COMPARE UNIT ID

LL_HRTIM_COMPAREUNIT_2

Compare unit 2 identifier

LL_HRTIM_COMPAREUNIT_4

Compare unit 4 identifier

CURRENT PUSH-PULL STATUS

LL_HRTIM_CPPSTAT_OUTPUT1

Signal applied on output 1 and output 2 forced inactive

LL_HRTIM_CPPSTAT_OUTPUT2

Signal applied on output 2 and output 1 forced inactive

DAC TRIGGER

LL_HRTIM_DACTRIG_NONE

No DAC synchronization event generated

LL_HRTIM_DACTRIG_DACTRIGOUT_1

DAC synchronization event generated on DACTrigOut1 output upon timer update

LL_HRTIM_DACTRIG_DACTRIGOUT_2

DAC synchronization event generated on DACTrigOut2 output upon timer update

LL_HRTIM_DACTRIG_DACTRIGOUT_3

DAC synchronization event generated on DACTrigOut3 output upon timer update

DLL CALIBRATION MODE

LL_HRTIM_DLLCALIBRATION_MODE_SINGLESHOT

Calibration is perfomed only once

LL_HRTIM_DLLCALIBRATION_MODE_CONTINUOUS

Calibration is performed periodically

DELAYED PROTECTION (DLYPRT) MODE

LL_HRTIM_DLYPRT_DELAYOUT1_EEV6

Timers A, B, C: Output 1 delayed Idle on external Event 6

LL_HRTIM_DLYPRT_DELAYOUT2_EEV6

Timers A, B, C: Output 2 delayed Idle on external Event 6

LL_HRTIM_DLYPRT_DELAYBOTH_EEV6

Timers A, B, C: Output 1 and output 2 delayed Idle on external Event 6

LL_HRTIM_DLYPRT_BALANCED_EEV6

Timers A, B, C: Balanced Idle on external Event 6

LL_HRTIM_DLYPRT_DELAYOUT1_EEV7

Timers A, B, C: Output 1 delayed Idle on external Event 7

LL_HRTIM_DLYPRT_DELAYOUT2_EEV7

Timers A, B, C: Output 2 delayed Idle on external Event 7

LL_HRTIM_DLYPRT_DELAYBOTH_EEV7

Timers A, B, C: Output 1 and output2 delayed Idle on external Event 7

LL_HRTIM_DLYPRT_BALANCED_EEV7

Timers A, B, C: Balanced Idle on external Event 7

LL_HRTIM_DLYPRT_DELAYOUT1_EEV8

Timers D, E: Output 1 delayed Idle on external Event 8

LL_HRTIM_DLYPRT_DELAYOUT2_EEV8

Timers D, E: Output 2 delayed Idle on external Event 8

LL_HRTIM_DLYPRT_DELAYBOTH_EEV8

Timers D, E: Output 1 and output 2 delayed Idle on external Event 8

LL_HRTIM_DLYPRT_BALANCED_EEV8

Timers D, E: Balanced Idle on external Event 8

LL_HRTIM_DLYPRT_DELAYOUT1_EEV9

Timers D, E: Output 1 delayed Idle on external Event 9

LL_HRTIM_DLYPRT_DELAYOUT2_EEV9

Timers D, E: Output 2 delayed Idle on external Event 9

LL_HRTIM_DLYPRT_DELAYBOTH_EEV9

Timers D, E: Output 1 and output2 delayed Idle on external Event 9

LL_HRTIM_DLYPRT_BALANCED_EEV9

Timers D, E: Balanced Idle on external Event 9

DEADTIME FALLING SIGN

LL_HRTIM_DT_FALLING_POSITIVE

Positive deadtime on falling edge

LL_HRTIM_DT_FALLING_NEGATIVE

Negative deadtime on falling edge

DEADTIME PRESCALER

LL_HRTIM_DT_PRESCALER_MUL8

fDTG = fHRTIM * 8

LL_HRTIM_DT_PRESCALER_MUL4

fDTG = fHRTIM * 4

LL_HRTIM_DT_PRESCALER_MUL2

fDTG = fHRTIM * 2

LL_HRTIM_DT_PRESCALER_DIV1

fDTG = fHRTIM

LL_HRTIM_DT_PRESCALER_DIV2

fDTG = fHRTIM / 2

LL_HRTIM_DT_PRESCALER_DIV4

fDTG = fHRTIM / 4

LL_HRTIM_DT_PRESCALER_DIV8

fDTG = fHRTIM / 8

LL_HRTIM_DT_PRESCALER_DIV16

fDTG = fHRTIM / 16

DEADTIME RISING SIGN

LL_HRTIM_DT_RISING_POSITIVE

Positive deadtime on rising edge

LL_HRTIM_DT_RISING_NEGATIVE

Negative deadtime on rising edge

EXTERNAL EVENT A or B COUNTER

LL_HRTIM_EVENT_COUNTER_A

External Event A Counter

LL_HRTIM_EVENT_COUNTER_B

External Event B Counter

EXTERNAL EVENT A or B RESET MODE

LL_HRTIM_EVENT_COUNTERRSTMODE_UNCONDITIONAL

External Event counter is reset on each reset / roll-over event

LL_HRTIM_EVENT_COUNTERRSTMODE_CONDITIONAL

External Event counter is reset on each reset / roll-over event only if no event occurs during last counting period

EXTERNAL EVENT FAST MODE

LL_HRTIM_EE_FASTMODE_DISABLE

External Event is re-synchronized by the HRTIM logic before acting on outputs

LL_HRTIM_EE_FASTMODE_ENABLE

External Event is acting asynchronously on outputs (low latency mode)

EXTERNAL EVENT DIGITAL FILTER

LL_HRTIM_EE_FILTER_NONE

Filter disabled

LL_HRTIM_EE_FILTER_1

fSAMPLING = fHRTIM, N=2

LL_HRTIM_EE_FILTER_2

fSAMPLING = fHRTIM, N=4

LL_HRTIM_EE_FILTER_3

fSAMPLING = fHRTIM, N=8

LL_HRTIM_EE_FILTER_4

fSAMPLING = fEEVS/2, N=6

LL_HRTIM_EE_FILTER_5

fSAMPLING = fEEVS/2, N=8

LL_HRTIM_EE_FILTER_6

fSAMPLING = fEEVS/4, N=6

LL_HRTIM_EE_FILTER_7

fSAMPLING = fEEVS/4, N=8

LL_HRTIM_EE_FILTER_8

fSAMPLING = fEEVS/8, N=6

LL_HRTIM_EE_FILTER_9

fSAMPLING = fEEVS/8, N=8

LL_HRTIM_EE_FILTER_10

fSAMPLING = fEEVS/16, N=5

LL_HRTIM_EE_FILTER_11

fSAMPLING = fEEVS/16, N=6

LL_HRTIM_EE_FILTER_12

fSAMPLING = fEEVS/16, N=8

LL_HRTIM_EE_FILTER_13

fSAMPLING = fEEVS/32, N=5

LL_HRTIM_EE_FILTER_14

fSAMPLING = fEEVS/32, N=6

LL_HRTIM_EE_FILTER_15

fSAMPLING = fEEVS/32, N=8

EXTERNAL EVENT POLARITY

LL_HRTIM_EE_POLARITY_HIGH

External event is active high

LL_HRTIM_EE_POLARITY_LOW

External event is active low

EXTERNAL EVENT PRESCALER

LL_HRTIM_EE_PRESCALER_DIV1

fEEVS = fHRTIM

LL_HRTIM_EE_PRESCALER_DIV2

fEEVS = fHRTIM / 2

LL_HRTIM_EE_PRESCALER_DIV4

fEEVS = fHRTIM / 4

LL_HRTIM_EE_PRESCALER_DIV8

fEEVS = fHRTIM / 8

EXTERNAL EVENT SENSITIVITY

LL_HRTIM_EE_SENSITIVITY_LEVEL

External event is active on level

LL_HRTIM_EE_SENSITIVITY_RISINGEDGE

External event is active on Rising edge

LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE

External event is active on Falling edge

LL_HRTIM_EE_SENSITIVITY_BOTHEDGES

External event is active on Rising and Falling edges

EXTERNAL EVENT SOURCE

LL_HRTIM_EEV1SRC_GPIO

External event source 1 for External Event 1

LL_HRTIM_EEV2SRC_GPIO

External event source 1 for External Event 2

LL_HRTIM_EEV3SRC_GPIO

External event source 1 for External Event 3

LL_HRTIM_EEV4SRC_GPIO

External event source 1 for External Event 4

LL_HRTIM_EEV5SRC_GPIO

External event source 1 for External Event 5

LL_HRTIM_EEV6SRC_GPIO

External event source 1 for External Event 6

LL_HRTIM_EEV7SRC_GPIO

External event source 1 for External Event 7

LL_HRTIM_EEV8SRC_GPIO

External event source 1 for External Event 8

LL_HRTIM_EEV9SRC_GPIO

External event source 1 for External Event 9

LL_HRTIM_EEV10SRC_GPIO

External event source 1 for External Event 10

LL_HRTIM_EEV1SRC_COMP2_OUT

External event source 2 for External Event 1

LL_HRTIM_EEV2SRC_COMP4_OUT

External event source 2 for External Event 2

LL_HRTIM_EEV3SRC_COMP6_OUT

External event source 2 for External Event 3

LL_HRTIM_EEV4SRC_COMP1_OUT

External event source 2 for External Event 4

LL_HRTIM_EEV5SRC_COMP3_OUT

External event source 2 for External Event 5

LL_HRTIM_EEV6SRC_COMP2_OUT

External event source 2 for External Event 6

LL_HRTIM_EEV7SRC_COMP4_OUT

External event source 2 for External Event 7

LL_HRTIM_EEV8SRC_COMP6_OUT

External event source 2 for External Event 8

LL_HRTIM_EEV9SRC_COMP5_OUT

External event source 2 for External Event 9

LL_HRTIM_EEV10SRC_COMP7_OUT

External event source 2 for External Event 10

LL_HRTIM_EEV1SRC_TIM1_TRGO

External event source 3 for External Event 1

LL_HRTIM_EEV2SRC_TIM2_TRGO

External event source 3 for External Event 2

LL_HRTIM_EEV3SRC_TIM3_TRGO

External event source 3 for External Event 3

LL_HRTIM_EEV4SRC_COMP5_OUT

External event source 3 for External Event 4

LL_HRTIM_EEV5SRC_COMP7_OUT

External event source 3 for External Event 5

LL_HRTIM_EEV6SRC_COMP1_OUT

External event source 3 for External Event 6

LL_HRTIM_EEV7SRC_TIM7_TRGO

External event source 3 for External Event 7

LL_HRTIM_EEV8SRC_COMP3_OUT

External event source 3 for External Event 8

LL_HRTIM_EEV9SRC_TIM15_TRGO

External event source 3 for External Event 9

LL_HRTIM_EEV10SRC_TIM6_TRGO

External event source 3 for External Event 10

LL_HRTIM_EEV1SRC_ADC1_AWD1

External event source 4 for External Event 1

LL_HRTIM_EEV2SRC_ADC1_AWD2

External event source 4 for External Event 2

LL_HRTIM_EEV3SRC_ADC1_AWD3

External event source 4 for External Event 3

LL_HRTIM_EEV4SRC_ADC2_AWD1

External event source 4 for External Event 4

LL_HRTIM_EEV5SRC_ADC2_AWD2

External event source 4 for External Event 5

LL_HRTIM_EEV6SRC_ADC2_AWD3

External event source 4 for External Event 6

LL_HRTIM_EEV7SRC_ADC3_AWD1

External event source 4 for External Event 7

LL_HRTIM_EEV8SRC_ADC4_AWD1

External event source 4 for External Event 8

LL_HRTIM_EEV9SRC_COMP4_OUT

External event source 4 for External Event 9

LL_HRTIM_EEV10SRC_ADC5_AWD1

External event source 4 for External Event 10

EXTERNAL EVENT ID

LL_HRTIM_EVENT_1

External event channel 1 identifier

LL_HRTIM_EVENT_2

External event channel 2 identifier

LL_HRTIM_EVENT_3

External event channel 3 identifier

LL_HRTIM_EVENT_4

External event channel 4 identifier

LL_HRTIM_EVENT_5

External event channel 5 identifier

LL_HRTIM_EVENT_6

External event channel 6 identifier

LL_HRTIM_EVENT_7

External event channel 7 identifier

LL_HRTIM_EVENT_8

External event channel 8 identifier

LL_HRTIM_EVENT_9

External event channel 9 identifier

LL_HRTIM_EVENT_10

External event channel 10 identifier

FAULT ID

LL_HRTIM_FAULT_1

Fault channel 1 identifier

LL_HRTIM_FAULT_2

Fault channel 2 identifier

LL_HRTIM_FAULT_3

Fault channel 3 identifier

LL_HRTIM_FAULT_4

Fault channel 4 identifier

LL_HRTIM_FAULT_5

Fault channel 5 identifier

LL_HRTIM_FAULT_6

Fault channel 6 identifier

FAULT BLANKING Source

LL_HRTIM_FLT_BLANKING_RSTALIGNED

Fault blanking source is Reset-aligned

LL_HRTIM_FLT_BLANKING_MOVING

Fault blanking source is Moving window

FAULT DIGITAL FILTER

LL_HRTIM_FLT_FILTER_NONE

Filter disabled

LL_HRTIM_FLT_FILTER_1

fSAMPLING= fHRTIM, N=2

LL_HRTIM_FLT_FILTER_2

fSAMPLING= fHRTIM, N=4

LL_HRTIM_FLT_FILTER_3

fSAMPLING= fHRTIM, N=8

LL_HRTIM_FLT_FILTER_4

fSAMPLING= fFLTS/2, N=6

LL_HRTIM_FLT_FILTER_5

fSAMPLING= fFLTS/2, N=8

LL_HRTIM_FLT_FILTER_6

fSAMPLING= fFLTS/4, N=6

LL_HRTIM_FLT_FILTER_7

fSAMPLING= fFLTS/4, N=8

LL_HRTIM_FLT_FILTER_8

fSAMPLING= fFLTS/8, N=6

LL_HRTIM_FLT_FILTER_9

fSAMPLING= fFLTS/8, N=8

LL_HRTIM_FLT_FILTER_10

fSAMPLING= fFLTS/16, N=5

LL_HRTIM_FLT_FILTER_11

fSAMPLING= fFLTS/16, N=6

LL_HRTIM_FLT_FILTER_12

fSAMPLING= fFLTS/16, N=8

LL_HRTIM_FLT_FILTER_13

fSAMPLING= fFLTS/32, N=5

LL_HRTIM_FLT_FILTER_14

fSAMPLING= fFLTS/32, N=6

LL_HRTIM_FLT_FILTER_15

fSAMPLING= fFLTS/32, N=8

FAULT POLARITY

LL_HRTIM_FLT_POLARITY_LOW

Fault input is active low

LL_HRTIM_FLT_POLARITY_HIGH

Fault input is active high

BURST FAULT PRESCALER

LL_HRTIM_FLT_PRESCALER_DIV1

fFLTS = fHRTIM

LL_HRTIM_FLT_PRESCALER_DIV2

fFLTS = fHRTIM / 2

LL_HRTIM_FLT_PRESCALER_DIV4

fFLTS = fHRTIM / 4

LL_HRTIM_FLT_PRESCALER_DIV8

fFLTS = fHRTIM / 8

FAULT Counter RESET Mode

LL_HRTIM_FLT_COUNTERRST_UNCONDITIONAL

Fault counter is reset on each reset / roll-over event

LL_HRTIM_FLT_COUNTERRST_CONDITIONAL

Fault counter is reset on each reset / roll-over event only if no fault occurred during last counting period.

FAULT SOURCE

LL_HRTIM_FLT_SRC_DIGITALINPUT

Fault input is FLT input pin

LL_HRTIM_FLT_SRC_INTERNAL

Fault input is FLT_Int signal (e.g. internal comparator)

LL_HRTIM_FLT_SRC_EEVINPUT

Fault input is external event pin

Get Flags Defines

LL_HRTIM_ISR_FLT1

LL_HRTIM_ISR_FLT2

LL_HRTIM_ISR_FLT3

LL_HRTIM_ISR_FLT4

LL_HRTIM_ISR_FLT5

LL_HRTIM_ISR_FLT6

LL_HRTIM_ISR_SYSFLT

LL_HRTIM_ISR_DLLRDY

LL_HRTIM_ISR_BMPER

LL_HRTIM_MISR_MCMP1

LL_HRTIM_MISR_MCMP2

LL_HRTIM_MISR_MCMP3

LL_HRTIM_MISR_MCMP4

LL_HRTIM_MISR_MREP

LL_HRTIM_MISR_SYNC

LL_HRTIM_MISR_MUPD

LL_HRTIM_TIMISR_CMP1

LL_HRTIM_TIMISR_CMP2

LL_HRTIM_TIMISR_CMP3

LL_HRTIM_TIMISR_CMP4

LL_HRTIM_TIMISR_REP

LL_HRTIM_TIMISR_UPD

LL_HRTIM_TIMISR_CPT1

LL_HRTIM_TIMISR_CPT2

LL_HRTIM_TIMISR_SET1

LL_HRTIM_TIMISR_RST1

LL_HRTIM_TIMISR_SET2

LL_HRTIM_TIMISR_RST2

LL_HRTIM_TIMISR_RST

LL_HRTIM_TIMISR_DLYPRT

HALF MODE

LL_HRTIM_HALF_MODE_DISABLED

HRTIM Half Mode is disabled

LL_HRTIM_HALF_MODE_ENABLE

HRTIM Half Mode is Half

INTLVD MODE

LL_HRTIM_INTERLEAVED_MODE_DISABLED

HRTIM interleaved Mode is disabled

LL_HRTIM_INTERLEAVED_MODE_DUAL

HRTIM interleaved Mode is Dual

LL_HRTIM_INTERLEAVED_MODE_TRIPLE

HRTIM interleaved Mode is Triple

LL_HRTIM_INTERLEAVED_MODE_QUAD

HRTIM interleaved Mode is Quad

IDLE PUSH-PULL STATUS

LL_HRTIM_IPPSTAT_OUTPUT1

Protection occurred when the output 1 was active and output 2 forced inactive

LL_HRTIM_IPPSTAT_OUTPUT2

Protection occurred when the output 2 was active and output 1 forced inactive

IT Defines

LL_HRTIM_IER_FLT1IE

LL_HRTIM_IER_FLT2IE

LL_HRTIM_IER_FLT3IE

LL_HRTIM_IER_FLT4IE

LL_HRTIM_IER_FLT5IE

LL_HRTIM_IER_FLT6IE

LL_HRTIM_IER_SYSFLTIE

LL_HRTIM_IER_DLLRDYIE

LL_HRTIM_IER_BMPERIE

LL_HRTIM_MDIER_MCMP1IE

LL_HRTIM_MDIER_MCMP2IE

LL_HRTIM_MDIER_MCMP3IE

LL_HRTIM_MDIER_MCMP4IE

LL_HRTIM_MDIER_MREPIE

LL_HRTIM_MDIER_SYNCIE

LL_HRTIM_MDIER_MUPDIE

LL_HRTIM_TIMDIER_CMP1IE

LL_HRTIM_TIMDIER_CMP2IE

LL_HRTIM_TIMDIER_CMP3IE

LL_HRTIM_TIMDIER_CMP4IE

LL_HRTIM_TIMDIER_REPIE

LL_HRTIM_TIMDIER_UPDIE

LL_HRTIM_TIMDIER_CPT1IE

LL_HRTIM_TIMDIER_CPT2IE

LL_HRTIM_TIMDIER_SET1IE

LL_HRTIM_TIMDIER_RST1IE

LL_HRTIM_TIMDIER_SET2IE

LL_HRTIM_TIMDIER_RST2IE

LL_HRTIM_TIMDIER_RSTIE

LL_HRTIM_TIMDIER_DLYPRTIE

COUNTER MODE

LL_HRTIM_MODE_CONTINUOUS

The timer operates in continuous (free-running) mode

LL_HRTIM_MODE_SINGLESHOT

The timer operates in non retriggerable single-shot mode

LL_HRTIM_MODE_RETRIGGERABLE

The timer operates in retriggerable single-shot mode

OUTPUT ID

LL_HRTIM_OUTPUT_TA1

Timer A - Output 1 identifier

LL_HRTIM_OUTPUT_TA2

Timer A - Output 2 identifier

LL_HRTIM_OUTPUT_TB1

Timer B - Output 1 identifier

LL_HRTIM_OUTPUT_TB2

Timer B - Output 2 identifier

LL_HRTIM_OUTPUT_TC1

Timer C - Output 1 identifier

LL_HRTIM_OUTPUT_TC2

Timer C - Output 2 identifier

LL_HRTIM_OUTPUT_TD1

Timer D - Output 1 identifier

LL_HRTIM_OUTPUT_TD2

Timer D - Output 2 identifier

LL_HRTIM_OUTPUT_TE1

Timer E - Output 1 identifier

LL_HRTIM_OUTPUT_TE2

Timer E - Output 2 identifier

LL_HRTIM_OUTPUT_TF1

Timer F - Output 1 identifier

LL_HRTIM_OUTPUT_TF2

Timer F - Output 2 identifier

OUTPUTSET INPUT

LL_HRTIM_OUTPUTSET_NONE

Reset the output set crossbar

LL_HRTIM_OUTPUTSET_RESYNC

Timer reset event coming solely from software or SYNC input forces an output level transision

LL_HRTIM_OUTPUTSET_TIMPER

Timer period event forces an output level transision

LL_HRTIM_OUTPUTSET_TIMCMP1

Timer compare 1 event forces an output level transision

LL_HRTIM_OUTPUTSET_TIMCMP2

Timer compare 2 event forces an output level transision

LL_HRTIM_OUTPUTSET_TIMCMP3

Timer compare 3 event forces an output level transision

LL_HRTIM_OUTPUTSET_TIMCMP4

Timer compare 4 event forces an output level transision

LL_HRTIM_OUTPUTSET_MASTERPER

The master timer period event forces an output level transision

LL_HRTIM_OUTPUTSET_MASTERCMP1

Master Timer compare 1 event forces an output level transision

LL_HRTIM_OUTPUTSET_MASTERCMP2

Master Timer compare 2 event forces an output level transision

LL_HRTIM_OUTPUTSET_MASTERCMP3

Master Timer compare 3 event forces an output level transision

LL_HRTIM_OUTPUTSET_MASTERCMP4

Master Timer compare 4 event forces an output level transision

LL_HRTIM_OUTPUTSET_TIMAEV1_TIMBCMP1

Timer event 1 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV2_TIMBCMP2

Timer event 2 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV3_TIMFCMP4

Timer event 3 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV4_TIMCCMP2

Timer event 4 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV5_TIMCCMP3

Timer event 5 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV6_TIMDCMP1

Timer event 6 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV7_TIMDCMP2

Timer event 7 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV8_TIMECMP3

Timer event 8 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMAEV9_TIMECMP4

Timer event 9 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV1_TIMACMP1

Timer event 1 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV2_TIMACMP2

Timer event 2 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV3_TIMFCMP3

Timer event 3 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV4_TIMCCMP3

Timer event 4 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV5_TIMCCMP4

Timer event 5 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV6_TIMDCMP3

Timer event 6 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV7_TIMDCMP4

Timer event 7 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV8_TIMECMP1

Timer event 8 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMBEV9_TIMECMP2

Timer event 9 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV1_TIMACMP2

Timer event 1 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV2_TIMACMP3

Timer event 2 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV3_TIMBCMP2

Timer event 3 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV4_TIMBCMP3

Timer event 4 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV5_TIMDCMP2

Timer event 5 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV6_TIMDCMP4

Timer event 6 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV7_TIMFCMP2

Timer event 7 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV8_TIMECMP3

Timer event 8 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMCEV9_TIMECMP4

Timer event 9 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV1_TIMACMP1

Timer event 1 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV2_TIMACMP4

Timer event 2 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV3_TIMBCMP2

Timer event 3 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV4_TIMBCMP4

Timer event 4 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV5_TIMFCMP1

Timer event 5 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV6_TIMFCMP3

Timer event 6 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV7_TIMCCMP4

Timer event 7 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV8_TIMECMP1

Timer event 8 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMDEV9_TIMECMP4

Timer event 9 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV1_TIMFCMP3

Timer event 1 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV2_TIMACMP4

Timer event 2 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV3_TIMBCMP3

Timer event 3 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV4_TIMBCMP4

Timer event 4 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV5_TIMCCMP1

Timer event 5 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV6_TIMCCMP2

Timer event 6 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV7_TIMDCMP1

Timer event 7 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV8_TIMDCMP2

Timer event 8 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMEEV9_TIMFCMP4

Timer event 9 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV1_TIMACMP3

Timer event 1 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV2_TIMBCMP1

Timer event 2 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV3_TIMBCMP4

Timer event 3 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV4_TIMCCMP1

Timer event 4 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV5_TIMCCMP4

Timer event 5 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV6_TIMDCMP3

Timer event 6 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV7_TIMDCMP4

Timer event 7 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV8_TIMECMP2

Timer event 8 forces the output to its active state

LL_HRTIM_OUTPUTSET_TIMFEV9_TIMECMP3

Timer event 9 forces the output to its active state

LL_HRTIM_OUTPUTSET_EEV_1

External event 1 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_2

External event 2 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_3

External event 3 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_4

External event 4 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_5

External event 5 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_6

External event 6 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_7

External event 7 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_8

External event 8 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_9

External event 9 forces an output level transision

LL_HRTIM_OUTPUTSET_EEV_10

External event 10 forces an output level transision

LL_HRTIM_OUTPUTSET_UPDATE

Timer register update event forces an output level transision

OUTPUT STATE

LL_HRTIM_OUTPUTSTATE_IDLE

Main operating mode, where the output can take the active or inactive level as programmed in the crossbar unit

LL_HRTIM_OUTPUTSTATE_RUN

Default operating state (e.g. after an HRTIM reset, when the outputs are disabled by software or during a burst mode operation)

LL_HRTIM_OUTPUTSTATE_FAULT

Safety state, entered in case of a shut-down request on FAULTx inputs

OUTPUT BURST MODE ENTRY MODE

LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR

The programmed Idle state is applied immediately to the Output

LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Deadtime is inserted on output before entering the idle mode

OUTPUT CHOPPER MODE

LL_HRTIM_OUT_CHOPPERMODE_DISABLED

Output signal is not altered

LL_HRTIM_OUT_CHOPPERMODE_ENABLED

Output signal is chopped by a carrier signal

OUTPUT FAULT STATE

LL_HRTIM_OUT_FAULTSTATE_NO_ACTION

The output is not affected by the fault input

LL_HRTIM_OUT_FAULTSTATE_ACTIVE

Output at active level when in FAULT state

LL_HRTIM_OUT_FAULTSTATE_INACTIVE

Output at inactive level when in FAULT state

LL_HRTIM_OUT_FAULTSTATE_HIGHZ

Output is tri-stated when in FAULT state

OUTPUT IDLE LEVEL

LL_HRTIM_OUT_IDLELEVEL_INACTIVE

Output at inactive level when in IDLE state

LL_HRTIM_OUT_IDLELEVEL_ACTIVE

Output at active level when in IDLE state

OUTPUT IDLE MODE

LL_HRTIM_OUT_NO_IDLE

The output is not affected by the burst mode operation

LL_HRTIM_OUT_IDLE_WHEN_BURST

The output is in idle state when requested by the burst mode controller

OUTPUT LEVEL

LL_HRTIM_OUT_LEVEL_INACTIVE

Corresponds to a logic level 0 for a positive polarity (High) and to a logic level 1 for a negative polarity (Low)

LL_HRTIM_OUT_LEVEL_ACTIVE

Corresponds to a logic level 1 for a positive polarity (High) and to a logic level 0 for a negative polarity (Low)

OUPUT_POLARITY

LL_HRTIM_OUT_POSITIVE_POLARITY

Output is acitve HIGH

LL_HRTIM_OUT_NEGATIVE_POLARITY

Output is active LOW

PRESCALER RATIO

LL_HRTIM_PRESCALERRATIO_MUL32

fHRCK: fHRTIM x 32 = 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_MUL16

fHRCK: fHRTIM x 16 = 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_MUL8

fHRCK: fHRTIM x 8 = 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_MUL4

fHRCK: fHRTIM x 4 = 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_MUL2

fHRCK: fHRTIM x 2 = 288 MHz - Resolution: 3.47 ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_DIV1

fHRCK: fHRTIM = 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_DIV2

fHRCK: fHRTIM / 2 = 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_DIV4

fHRCK: fHRTIM / 4 = 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

RESET TRIGGER

LL_HRTIM_RESETTRIG_NONE

No counter reset trigger

LL_HRTIM_RESETTRIG_UPDATE

The timer counter is reset upon update event

LL_HRTIM_RESETTRIG_CMP2

The timer counter is reset upon Timer Compare 2 event

LL_HRTIM_RESETTRIG_CMP4

The timer counter is reset upon Timer Compare 4 event

LL_HRTIM_RESETTRIG_MASTER_PER

The timer counter is reset upon master timer period event

LL_HRTIM_RESETTRIG_MASTER_CMP1

The timer counter is reset upon master timer Compare 1 event

LL_HRTIM_RESETTRIG_MASTER_CMP2

The timer counter is reset upon master timer Compare 2 event

LL_HRTIM_RESETTRIG_MASTER_CMP3

The timer counter is reset upon master timer Compare 3 event

LL_HRTIM_RESETTRIG_MASTER_CMP4

The timer counter is reset upon master timer Compare 4 event

LL_HRTIM_RESETTRIG_EEV_1

The timer counter is reset upon external event 1

LL_HRTIM_RESETTRIG_EEV_2

The timer counter is reset upon external event 2

LL_HRTIM_RESETTRIG_EEV_3

The timer counter is reset upon external event 3

LL_HRTIM_RESETTRIG_EEV_4

The timer counter is reset upon external event 4

LL_HRTIM_RESETTRIG_EEV_5

The timer counter is reset upon external event 5

LL_HRTIM_RESETTRIG_EEV_6

The timer counter is reset upon external event 6

LL_HRTIM_RESETTRIG_EEV_7

The timer counter is reset upon external event 7

LL_HRTIM_RESETTRIG_EEV_8

The timer counter is reset upon external event 8

LL_HRTIM_RESETTRIG_EEV_9

The timer counter is reset upon external event 9

LL_HRTIM_RESETTRIG_EEV_10

The timer counter is reset upon external event 10

LL_HRTIM_RESETTRIG_OTHER1_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER1_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER1_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER2_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER2_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER2_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER3_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER3_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER3_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER4_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER4_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER4_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER5_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER5_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER5_CMP4

The timer counter is reset upon other timer Compare 4 event

SYNCHRONIZATION INPUT SOURCE

LL_HRTIM_SYNCIN_SRC_NONE

HRTIM is not synchronized and runs in standalone mode

LL_HRTIM_SYNCIN_SRC_TIM_EVENT

The HRTIM is synchronized with the on-chip timer

LL_HRTIM_SYNCIN_SRC_EXTERNAL_EVENT

A positive pulse on SYNCIN input triggers the HRTIM

SYNCHRONIZATION OUTPUT POLARITY

LL_HRTIM_SYNCOUT_DISABLED

Synchronization output event is disabled

LL_HRTIM_SYNCOUT_POSITIVE_PULSE

SCOUT pin has a low idle level and issues a positive pulse of 16 fHRTIM clock cycles length for the synchronization

LL_HRTIM_SYNCOUT_NEGATIVE_PULSE

SCOUT pin has a high idle level and issues a negative pulse of 16 fHRTIM clock cycles length for the synchronization

SYNCHRONIZATION OUTPUT SOURCE

LL_HRTIM_SYNCOUT_SRC_MASTER_START

A pulse is sent on the SYNCOUT output upon master timer start event

LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1

A pulse is sent on the SYNCOUT output upon master timer compare 1 event

LL_HRTIM_SYNCOUT_SRC_TIMA_START

A pulse is sent on the SYNCOUT output upon timer A start or reset events

LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

A pulse is sent on the SYNCOUT output upon timer A compare 1 event

TIMER ID

LL_HRTIM_TIMER_NONE

Master timer identifier

LL_HRTIM_TIMER_MASTER

Master timer identifier

LL_HRTIM_TIMER_A

Timer A identifier

LL_HRTIM_TIMER_B

Timer B identifier

LL_HRTIM_TIMER_C

Timer C identifier

LL_HRTIM_TIMER_D

Timer D identifier

LL_HRTIM_TIMER_E

Timer E identifier

LL_HRTIM_TIMER_F

Timer F identifier

LL_HRTIM_TIMER_X

LL_HRTIM_TIMER_ALL

TIMER EXTERNAL EVENT FILTER

LL_HRTIM_EEFLTR_NONE

LL_HRTIM_EEFLTR_BLANKINGCMP1

Blanking from counter reset/roll-over to Compare 1U

LL_HRTIM_EEFLTR_BLANKINGCMP2

Blanking from counter reset/roll-over to Compare 2U

LL_HRTIM_EEFLTR_BLANKINGCMP3

Blanking from counter reset/roll-over to Compare 3U

LL_HRTIM_EEFLTR_BLANKINGCMP4

Blanking from counter reset/roll-over to Compare 4U

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF1_TIMBCMP1

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF2_TIMBCMP4

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF3_TIMBOUT2

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF4_TIMCCMP1

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF5_TIMCCMP4

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF6_TIMFCMP1

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF7_TIMDCMP1

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKING_TIMAEEF8_TIMECMP2

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF1_TIMACMP1

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF2_TIMACMP4

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF3_TIMAOUT2

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF4_TIMCCMP1

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF5_TIMCCMP2

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF6_TIMFCMP2

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF7_TIMDCMP2

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKING_TIMBEEF8_TIMECMP1

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF1_TIMACMP2

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF2_TIMBCMP1

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF3_TIMBCMP4

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF4_TIMFCMP1

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF5_TIMDCMP1

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF6_TIMDCMP4

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF7_TIMDOUT2

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKING_TIMCEEF8_TIMECMP4

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF1_TIMACMP1

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF2_TIMBCMP2

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF3_TIMCCMP1

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF4_TIMCCMP2

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF5_TIMCOUT2

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF6_TIMECMP1

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF7_TIMECMP4

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKING_TIMDEEF8_TIMFCMP4

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF1_TIMACMP2

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF2_TIMBCMP1

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF3_TIMCCMP1

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF4_TIMFCMP4

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF5_TIMFOUT2

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF6_TIMDCMP1

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF7_TIMDCMP4

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKING_TIMEEEF8_TIMDOUT2

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF1_TIMACMP4

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF2_TIMBCMP2

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF3_TIMCCMP4

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF4_TIMDCMP2

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF5_TIMDCMP4

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF6_TIMECMP1

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF7_TIMECMP4

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKING_TIMFEEF8_TIMEOUT2

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_WINDOWINGCMP2

Windowing from counter reset/roll-over to Compare 2U

LL_HRTIM_EEFLTR_WINDOWINGCMP3

Windowing from counter reset/roll-over to Compare 3U

LL_HRTIM_EEFLTR_WINDOWINGTIM

Windowing from another timing unit: TIMWIN source

TIMER EXTERNAL EVENT LATCH STATUS

LL_HRTIM_EELATCH_DISABLED

Event is ignored if it happens during a blank, or passed through during a window

LL_HRTIM_EELATCH_ENABLED

Event is latched and delayed till the end of the blanking or windowing period

UPDATE GATING

LL_HRTIM_UPDATEGATING_INDEPENDENT

Update done independently from the DMA burst transfer completion

LL_HRTIM_UPDATEGATING_DMABURST

Update done when the DMA burst transfer is completed

LL_HRTIM_UPDATEGATING_DMABURST_UPDATE

Update done on timer roll-over following a DMA burst transfer completion

LL_HRTIM_UPDATEGATING_UPDEN1

Slave timer only - Update done on a rising edge of HRTIM update enable input 1

LL_HRTIM_UPDATEGATING_UPDEN2

Slave timer only - Update done on a rising edge of HRTIM update enable input 2

LL_HRTIM_UPDATEGATING_UPDEN3

Slave timer only - Update done on a rising edge of HRTIM update enable input 3

LL_HRTIM_UPDATEGATING_UPDEN1_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 1

LL_HRTIM_UPDATEGATING_UPDEN2_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 2

LL_HRTIM_UPDATEGATING_UPDEN3_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 3

UPDATE TRIGGER

LL_HRTIM_UPDATETRIG_NONE

Register update is disabled

LL_HRTIM_UPDATETRIG_MASTER

Register update is triggered by the master timer update

LL_HRTIM_UPDATETRIG_TIMER_A

Register update is triggered by the timer A update

LL_HRTIM_UPDATETRIG_TIMER_B

Register update is triggered by the timer B update

LL_HRTIM_UPDATETRIG_TIMER_C

Register update is triggered by the timer C update

LL_HRTIM_UPDATETRIG_TIMER_D

Register update is triggered by the timer D update

LL_HRTIM_UPDATETRIG_TIMER_E

Register update is triggered by the timer E update

LL_HRTIM_UPDATETRIG_TIMER_F

Register update is triggered by the timer F update

LL_HRTIM_UPDATETRIG_REPETITION

Register update is triggered when the counter rolls over and HRTIM_REPx = 0

LL_HRTIM_UPDATETRIG_RESET

Register update is triggered by counter reset or roll-over to 0 after reaching the period value in continuous mode

Exported_Macros

__LL_HRTIM_GET_OUTPUT_STATE

Description:

- HELPER macro returning the output state from output enable/disable status.

Parameters:

- __OUTPUT_STATUS_EN__: output enable status
- __OUTPUT_STATUS_DIS__: output Disable status

Return value:

- Returned: value can be one of the following values:
  - LL_HRTIM_OUTPUTSTATE_IDLE
  - LL_HRTIM_OUTPUTSTATE_RUN
  - LL_HRTIM_OUTPUTSTATE_FAULT

Common Write and read registers Macros

LL_HRTIM_WriteReg

Description:

- Write a value in HRTIM register.

Parameters:

- __INSTANCE__: HRTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_HRTIM_ReadReg

Description:

- Read a value in HRTIM register.

Parameters:

- __INSTANCE__: HRTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

79 LL I2C Generic Driver

79.1 I2C Firmware driver registers structures

79.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32g4xx_ll_i2c.h

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t DigitalFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***

  Specifies the peripheral mode. This parameter can be a value of ***I2C_LL_EC_PERIPHERAL_MODE***This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::Timing***

  Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **__LL_I2C_CONVERT_TIMINGS()**This feature can be modified afterwards using unitary function **LL_I2C_SetTiming()**.
- ***uint32_t LL_I2C_InitTypeDef::AnalogFilter***

  Enables or disables analog noise filter. This parameter can be a value of ***I2C_LL_EC_ANALOGFILTER_SELECTION***This feature can be modified afterwards using unitary functions **LL_I2C_EnableAnalogFilter()** or **LL_I2C_DisableAnalogFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::DigitalFilter***

  Configures the digital noise filter. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x0FThis feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***

  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FFThis feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***

  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of ***I2C_LL_EC_I2C_ACKNOWLEDGE***This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***

  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of ***I2C_LL_EC_OWNADDRESS1***This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

79.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

79.2.1 Detailed description of functions

LL_I2C_Enable

Function name

__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)

Function description

Enable I2C peripheral (PE = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name

__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)

Function description

Disable I2C peripheral (PE = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabled (I2C_TypeDef * I2Cx)

Function description

Check if the I2C peripheral is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_IsEnabled

LL_I2C_ConfigFilters

Function name

__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)

Function description

Configure Noise Filters (Analog and Digital).

Parameters

- **I2Cx:** I2C Instance.
- **AnalogFilter:** This parameter can be one of the following values:
  - LL_I2C_ANALOGFILTER_ENABLE
  - LL_I2C_ANALOGFILTER_DISABLE
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_ConfigFilters
- CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name

__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)

Function description

Configure Digital Noise Filter.

Parameters

- **I2Cx:** I2C Instance.
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (I2C_TypeDef * I2Cx)

Function description

Get the current Digital Noise Filter configuration.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Enable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Disable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Check if Analog Noise Filter is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Enable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL_I2C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Disable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Check if DMA transmission requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Enable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Disable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

LL_I2C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Check if DMA reception requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I2Cx:** I2C Instance
- **Direction:** This parameter can be one of the following values:
  - LL_I2C_DMA_REG_DATA_TRANSMIT
  - LL_I2C_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_DMA_GetRegAddr
- RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name

__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)

Function description

Enable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name

__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)

Function description

Disable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (I2C_TypeDef * I2Cx)

Function description

Check if Clock stretching is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Enable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Disable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Check if hardware byte control in slave mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Enable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
- This bit can only be programmed when Digital Filter is disabled.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Disable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Check if Wakeup from STOP is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Enable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When enabled the Address 0x00 is ACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Disable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When disabled the Address 0x00 is NACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (I2C_TypeDef * I2Cx)

Function description

Check if General Call is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_IsEnabledGeneralCall

LL_I2C_SetMasterAddressingMode

Function name

__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)

Function description

Configure the Master to operate in 7-bit or 10-bit addressing mode.

Parameters

- **I2Cx:** I2C Instance.
- **AddressingMode:** This parameter can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Return values

- **None:**

Notes

- Changing this bit is not allowed, when the START bit is set.

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (I2C_TypeDef * I2Cx)

Function description

Get the Master addressing mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)

Function description

Set the Own Address1.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF.
- **OwnAddrSize:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS1_7BIT
  - LL_I2C_OWNADDRESS1_10BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1 LL_I2C_SetOwnAddress1
- OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)

Function description

Set the 7bits Own Address2.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F.
- **OwnAddrMask:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS2_NOMASK
  - LL_I2C_OWNADDRESS2_MASK01
  - LL_I2C_OWNADDRESS2_MASK02
  - LL_I2C_OWNADDRESS2_MASK03
  - LL_I2C_OWNADDRESS2_MASK04
  - LL_I2C_OWNADDRESS2_MASK05
  - LL_I2C_OWNADDRESS2_MASK06
  - LL_I2C_OWNADDRESS2_MASK07

Return values

- **None:**

Notes

- This action has no effect if own address2 is enabled.

Reference Manual to LL API cross reference:

- OAR2 OA2 LL_I2C_SetOwnAddress2
- OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name

__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)

Function description

Configure the SDA setup, hold time and the SCL high, low period.

Parameters

- **I2Cx:** I2C Instance.
- **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (I2C_TypeDef * I2Cx)

Function description

Get the Timing Prescaler setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (I2C_TypeDef * I2Cx)

Function description

Get the SCL low period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (I2C_TypeDef * I2Cx)

Function description

Get the SCL high period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (I2C_TypeDef * I2Cx)

Function description

Get the SDA hold time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (I2C_TypeDef * I2Cx)

Function description

Get the SDA setup time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name

__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I2Cx:** I2C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_SetMode
- CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)

Function description

Get peripheral mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_GetMode
- CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Enable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Disable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Enable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Disable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)

Function description

Configure the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
- **TimeoutB:**

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/ orTimeoutB).

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)

Function description

Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)

Function description

Set the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This bit can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_SetSMBusTimeoutAMode

LL_I2C_GetSMBusTimeoutAMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)

Function description

Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutB is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB

LL_I2C_GetSMBusTimeoutB

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Extented Cumulative Clock TimeoutB setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Enable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Disable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Check if the SMBus Clock Timeout is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_IsEnabledSMBusTimeout

LL_I2C_EnableIT_TX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)

Function description

Enable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)

Function description

Disable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (I2C_TypeDef * I2Cx)

Function description

Check if the TXIS Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)

Function description

Enable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)

Function description

Disable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (I2C_TypeDef * I2Cx)

Function description

Check if the RXNE Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_EnableIT_ADDR

LL_I2C_DisableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_DisableIT_ADDR

LL_I2C_IsEnabledIT_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Check if Address match interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_IsEnabledIT_ADDR

LL_I2C_EnableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_EnableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Enable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_EnableIT_NACK

LL_I2C_DisableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_DisableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Disable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_DisableIT_NACK

LL_I2C_IsEnabledIT_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_NACK (I2C_TypeDef * I2Cx)

Function description

Check if Not acknowledge received interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_IsEnabledIT_NACK

LL_I2C_EnableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Enable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_EnableIT_STOP

LL_I2C_DisableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Disable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_DisableIT_STOP

LL_I2C_IsEnabledIT_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (I2C_TypeDef * I2Cx)

Function description

Check if STOP detection interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)

Function description

Enable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)

Function description

Disable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (I2C_TypeDef * I2Cx)

Function description

Check if Transfer Complete interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Enable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Disable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (I2C_TypeDef * I2Cx)

Function description

Check if Error interrupts are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit data register empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit interrupt flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Receive data register not empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.

Reference Manual to LL API cross reference:

- ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Address matched flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.

Reference Manual to LL API cross reference:

- ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Not Acknowledge received flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a NACK is received after a byte transmission.

Reference Manual to LL API cross reference:

- ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When arbitration lost.

Reference Manual to LL API cross reference:

- ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Overrun/Underrun flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).

Reference Manual to LL API cross reference:

- ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus PEC error flag in reception.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.

Reference Manual to LL API cross reference:

- ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.

Reference Manual to LL API cross reference:

- ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.

Reference Manual to LL API cross reference:

- ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus Busy flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Start condition is detected.

Reference Manual to LL API cross reference:

- ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Clear Address Matched flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Clear Not Acknowledge flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Clear Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Clear Transmit data register empty flag (TXE).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can be clear by software in order to flush the transmit data register (TXDR).

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Clear Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Clear Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Clear Overrun/Underrun flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Clear SMBus PEC error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR PECCF LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR TIMOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Enable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Automatic end mode : a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Disable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Software end mode : TC flag is set when NBYTES data are transferre, stretching SCL low.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Check if automatic STOP condition is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name

__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)

Function description

Enable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name

__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)

Function description

Disable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (I2C_TypeDef * I2Cx)

Function description

Check if reload mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name

__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)

Function description

Configure the number of bytes for transfer.

Parameters

- **I2Cx:** I2C Instance.
- **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (I2C_TypeDef * I2Cx)

Function description

Get the number of bytes configured for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name

__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)

Function description

Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.

Parameters

- **I2Cx:** I2C Instance.
- **TypeAcknowledge:** This parameter can be one of the following values:
  - LL_I2C_ACK
  - LL_I2C_NACK

Return values

- **None:**

Notes

- Usage in Slave mode only.

Reference Manual to LL API cross reference:

- CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)

Function description

Generate a START or RESTART condition.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.

Reference Manual to LL API cross reference:

- CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)

Function description

Generate a STOP condition after the current byte transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Enable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Disable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master only sends the first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name

__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)

Function description

Configure the transfer direction (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **TransferRequest:** This parameter can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (I2C_TypeDef * I2Cx)

Function description

Get the transfer direction requested (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name

__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)

Function description

Configure the slave address for transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (I2C_TypeDef * I2Cx)

Function description

Get the slave address programmed for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name

__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)

Function description

Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** Specifies the slave address to be programmed.
- **SlaveAddrSize:** This parameter can be one of the following values:
  - LL_I2C_ADDRSLAVE_7BIT
  - LL_I2C_ADDRSLAVE_10BIT
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255.
- **EndMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_RELOAD
  - LL_I2C_MODE_AUTOEND
  - LL_I2C_MODE_SOFTEND
  - LL_I2C_MODE_SMBUS_RELOAD
  - LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC
  - LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC
- **Request:** This parameter can be one of the following values:
  - LL_I2C_GENERATE_NOSTARTSTOP
  - LL_I2C_GENERATE_STOP
  - LL_I2C_GENERATE_START_READ
  - LL_I2C_GENERATE_START_WRITE
  - LL_I2C_GENERATE_RESTART_7BIT_READ
  - LL_I2C_GENERATE_RESTART_7BIT_WRITE
  - LL_I2C_GENERATE_RESTART_10BIT_READ
  - LL_I2C_GENERATE_RESTART_10BIT_WRITE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_HandleTransfer
- CR2 ADD10 LL_I2C_HandleTransfer
- CR2 RD_WRN LL_I2C_HandleTransfer
- CR2 START LL_I2C_HandleTransfer
- CR2 STOP LL_I2C_HandleTransfer
- CR2 RELOAD LL_I2C_HandleTransfer
- CR2 NBYTES LL_I2C_HandleTransfer
- CR2 AUTOEND LL_I2C_HandleTransfer
- CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (I2C_TypeDef * I2Cx)

Function description

Indicate the value of transfer direction (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_DIRECTION_WRITE
  - LL_I2C_DIRECTION_READ

Notes

- RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.

Reference Manual to LL API cross reference:

- ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (I2C_TypeDef * I2Cx)

Function description

Return the slave matched address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Check if the SMBus Packet Error byte internal comparison is requested or not.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Packet Error byte calculated.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- Macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)

Function description

Read Receive Data register.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name

__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)

Function description

Write in Transmit Data Register .

Parameters

- **I2Cx:** I2C Instance.
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name

ErrorStatus LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Initialize the I2C registers according to the specified parameters in I2C_InitStruct.

Parameters

- **I2Cx:** I2C Instance.
- **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are initialized
  - ERROR: Not applicable

LL_I2C_DeInit

Function name

ErrorStatus LL_I2C_DeInit (I2C_TypeDef * I2Cx)

Function description

De-initialize the I2C registers to their default reset values.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are de-initialized
  - ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name

void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Set each LL_I2C_InitTypeDef field to default value.

Parameters

- **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.

Return values

- **None:**

79.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

79.3.1 I2C

I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT

Master operates in 7-bit addressing mode.

LL_I2C_ADDRESSING_MODE_10BIT

Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT

Slave Address in 7-bit.

LL_I2C_ADDRSLAVE_10BIT

Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE

Analog filter is enabled.

LL_I2C_ANALOGFILTER_DISABLE

Analog filter is disabled.

Clear Flags Defines

LL_I2C_ICR_ADDRCF

Address Matched flag

LL_I2C_ICR_NACKCF

Not Acknowledge flag

LL_I2C_ICR_STOPCF

Stop detection flag

LL_I2C_ICR_BERRCF

Bus error flag

LL_I2C_ICR_ARLOCF

Arbitration Lost flag

LL_I2C_ICR_OVRCF

Overrun/Underrun flag

LL_I2C_ICR_PECCF

PEC error flag

LL_I2C_ICR_TIMOUTCF

Timeout detection flag

LL_I2C_ICR_ALERTCF

Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE

Write transfer request by master, slave enters receiver mode.

LL_I2C_DIRECTION_READ

Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_I2C_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP

Don't Generate Stop and Start condition.

LL_I2C_GENERATE_STOP

Generate Stop condition (Size should be set to 0).

LL_I2C_GENERATE_START_READ

Generate Start for read request.

LL_I2C_GENERATE_START_WRITE

Generate Start for write request.

LL_I2C_GENERATE_RESTART_7BIT_READ

Generate Restart for read request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_7BIT_WRITE

Generate Restart for write request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_10BIT_READ

Generate Restart for read request, slave 10Bit address.

LL_I2C_GENERATE_RESTART_10BIT_WRITE

Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE

Transmit data register empty

LL_I2C_ISR_TXIS

Transmit interrupt status

LL_I2C_ISR_RXNE

Receive data register not empty

LL_I2C_ISR_ADDR

Address matched (slave mode)

LL_I2C_ISR_NACKF

Not Acknowledge received flag

LL_I2C_ISR_STOPF

Stop detection flag

LL_I2C_ISR_TC

Transfer Complete (master mode)

LL_I2C_ISR_TCR

Transfer Complete Reload

LL_I2C_ISR_BERR

Bus error

LL_I2C_ISR_ARLO

Arbitration lost

LL_I2C_ISR_OVR

Overrun/Underrun (slave mode)

LL_I2C_ISR_PECERR

PEC Error in reception (SMBus mode)

LL_I2C_ISR_TIMEOUT

Timeout detection flag (SMBus mode)

LL_I2C_ISR_ALERT

SMBus alert (SMBus mode)

LL_I2C_ISR_BUSY

Bus busy

Acknowledge Generation

LL_I2C_ACK

ACK is sent after current received byte.

LL_I2C_NACK

NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE

TX Interrupt enable

LL_I2C_CR1_RXIE

RX Interrupt enable

LL_I2C_CR1_ADDRIE

Address match Interrupt enable (slave only)

LL_I2C_CR1_NACKIE

Not acknowledge received Interrupt enable

LL_I2C_CR1_STOPIE

STOP detection Interrupt enable

LL_I2C_CR1_TCIE

Transfer Complete interrupt enable

LL_I2C_CR1_ERRIE

Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD

Enable I2C Reload mode.

LL_I2C_MODE_AUTOEND

Enable I2C Automatic end mode with no HW PEC comparison.

LL_I2C_MODE_SOFTEND

Enable I2C Software end mode with no HW PEC comparison.

LL_I2C_MODE_SMBUS_RELOAD

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC

Enable SMBUS Software end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC

Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT

Own address 1 is a 7-bit address.

LL_I2C_OWNADDRESS1_10BIT

Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK

Own Address2 No mask.

LL_I2C_OWNADDRESS2_MASK01

Only Address2 bits[7:2] are compared.

LL_I2C_OWNADDRESS2_MASK02

Only Address2 bits[7:3] are compared.

LL_I2C_OWNADDRESS2_MASK03

Only Address2 bits[7:4] are compared.

LL_I2C_OWNADDRESS2_MASK04

Only Address2 bits[7:5] are compared.

LL_I2C_OWNADDRESS2_MASK05

Only Address2 bits[7:6] are compared.

LL_I2C_OWNADDRESS2_MASK06

Only Address2 bits[7] are compared.

LL_I2C_OWNADDRESS2_MASK07

No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C

I2C Master or Slave mode

LL_I2C_MODE_SMBUS_HOST

SMBus Host address acknowledge

LL_I2C_MODE_SMBUS_DEVICE

SMBus Device default mode (Default address not acknowledge)

LL_I2C_MODE_SMBUS_DEVICE_ARP

SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE

Master request a write transfer.

LL_I2C_REQUEST_READ

Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW

TimeoutA is used to detect SCL low level timeout.

LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA

TimeoutA enable bit

LL_I2C_SMBUS_TIMEOUTB

TimeoutB (extended clock) enable bit

LL_I2C_SMBUS_ALL_TIMEOUT

TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __DATA_SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __DATA_HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __CLOCK_HIGH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __CLOCK_LOW_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

80 LL IWDG Generic Driver

80.1 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

80.1.1 Detailed description of functions

LL_IWDG_Enable

Function name

__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)

Function description

Start the Independent Watchdog.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Notes

- Except if the hardware watchdog option is selected

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Reloads IWDG counter with value defined in the reload register.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_DisableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)

Function description

Select the prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (IWDG_TypeDef * IWDGx)

Function description

Get the selected prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)

Function description

Specify the IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance
- **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Get the specified IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name

__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)

Function description

Specify high limit of the window value to be compared to the down-counter.

Parameters

- **IWDGx:** IWDG Instance
- **Window:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetWindow (IWDG_TypeDef * IWDGx)

Function description

Get the high limit of the window value specified.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_GetWindow

LL_IWDG_IsActiveFlag_PVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (IWDG_TypeDef * IWDGx)

Function description

Check if flag Prescaler Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (IWDG_TypeDef * IWDGx)

Function description

Check if flag Reload Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsActiveFlag_WVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_WVU (IWDG_TypeDef * IWDGx)

Function description

Check if flag Window Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR WVU LL_IWDG_IsActiveFlag_WVU

LL_IWDG_IsReady

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsReady (IWDG_TypeDef * IWDGx)

Function description

Check if all flags Prescaler, Reload & Window Value Update are reset or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bits (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsReady
- SR WVU LL_IWDG_IsReady
- SR RVU LL_IWDG_IsReady

80.2 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

80.2.1 IWDG

IWDG

Get Flags Defines

LL_IWDG_SR_PVU

Watchdog prescaler value update

LL_IWDG_SR_RVU

Watchdog counter reload value update

LL_IWDG_SR_WVU

Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4

Divider by 4

LL_IWDG_PRESCALER_8

Divider by 8

LL_IWDG_PRESCALER_16

Divider by 16

LL_IWDG_PRESCALER_32

Divider by 32

LL_IWDG_PRESCALER_64

Divider by 64

LL_IWDG_PRESCALER_128

Divider by 128

LL_IWDG_PRESCALER_256

Divider by 256

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

81 LL LPTIM Generic Driver

81.1 LPTIM Firmware driver registers structures

81.1.1 LL_LPTIM_InitTypeDef

LL_LPTIM_InitTypeDef is defined in the stm32g4xx_ll_lptim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***

  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of ***LPTIM_LL_EC_CLK_SOURCE***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***

  Specifies the prescaler division ratio. This parameter can be a value of ***LPTIM_LL_EC_PRESCALER***.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***

  Specifies the waveform shape. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_WAVEFORM***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Polarity***

  Specifies waveform polarity. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_POLARITY***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.

81.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

81.2.1 Detailed description of functions

LL_LPTIM_DeInit

Function name

ErrorStatus LL_LPTIM_DeInit (LPTIM_TypeDef * LPTIMx)

Function description

Set LPTIMx registers to their reset values.

Parameters

- **LPTIMx:** LP Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx registers are de-initialized
  - ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name

void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Set each fields of the LPTIM_InitStruct structure to its default value.

Parameters

- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **None:**

LL_LPTIM_Init

Function name

ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Configure the LPTIMx peripheral according to the specified parameters.

Parameters

- **LPTIMx:** LP Timer Instance
- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx instance has been initialized
  - ERROR: LPTIMx instance hasn't been initialized

Notes

- LL_LPTIM_Init can only be called when the LPTIM instance is disabled.
- LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

LL_LPTIM_Disable

Function name

void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)

Function description

Disable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Disable

LL_LPTIM_Enable

Function name

__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)

Function description

Enable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Enable

LL_LPTIM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM instance is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name

__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)

Function description

Starts the LPTIM counter in the desired mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **OperatingMode:** This parameter can be one of the following values:
  - LL_LPTIM_OPERATING_MODE_CONTINUOUS
  - LL_LPTIM_OPERATING_MODE_ONESHOT

Return values

- **None:**

Notes

- LPTIM instance must be enabled before starting the counter.
- It is possible to change on the fly from One Shot mode to Continuous mode.

Reference Manual to LL API cross reference:

- CR CNTSTRT LL_LPTIM_StartCounter
- CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_EnableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_EnableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Enable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After calling this function any read access to LPTIM_CNT register will asynchronously reset the LPTIM_CNT register content.

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_EnableResetAfterRead

LL_LPTIM_DisableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_DisableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Disable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_DisableResetAfterRead

LL_LPTIM_IsEnabledResetAfterRead

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the reset after read feature is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_IsEnabledResetAfterRead

LL_LPTIM_ResetCounter

Function name

__STATIC_INLINE void LL_LPTIM_ResetCounter (LPTIM_TypeDef * LPTIMx)

Function description

Reset of the LPTIM_CNT counter register (synchronous).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- Due to the synchronous nature of this reset, it only takes place after a synchronization delay of 3 LPTIM core clock cycles (LPTIM core clock may be different from APB clock).
- COUNTRST is automatically cleared by hardware

Reference Manual to LL API cross reference:

- CR COUNTRST LL_LPTIM_ResetCounter
-

LL_LPTIM_SetUpdateMode

Function name

__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)

Function description

Set the LPTIM registers update mode (enable/disable register preload)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **UpdateMode:** This parameter can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (LPTIM_TypeDef * LPTIMx)

Function description

Get the LPTIM registers update mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name

__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)

Function description

Set the auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **AutoReload:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- The LPTIMx_ARR register content must only be modified when the LPTIM is enabled
- After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag is set, will lead to unpredictable results.
- autoreload value be strictly greater than the compare value.

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (LPTIM_TypeDef * LPTIMx)

Function description

Get actual auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **AutoReload:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetCompare

Function name

__STATIC_INLINE void LL_LPTIM_SetCompare (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CMP register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMPOK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CMP CMP LL_LPTIM_SetCompare

LL_LPTIM_GetCompare

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCompare (LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CMP CMP LL_LPTIM_GetCompare

LL_LPTIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (LPTIM_TypeDef * LPTIMx)

Function description

Get actual counter value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Counter:** value

Notes

- When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

Reference Manual to LL API cross reference:

- CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name

__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)

Function description

Set the counter mode (selection of the LPTIM counter clock source).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Return values

- **None:**

Notes

- The counter mode can be set only when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (LPTIM_TypeDef * LPTIMx)

Function description

Get the counter mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_ConfigOutput

Function name

__STATIC_INLINE void LL_LPTIM_ConfigOutput (LPTIM_TypeDef * LPTIMx, uint32_t Waveform, uint32_t Polarity)

Function description

Configure the LPTIM instance output (LPTIMx_OUT)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- Regarding the LPTIM output polarity the change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_ConfigOutput
- CFGR WAVPOL LL_LPTIM_ConfigOutput

LL_LPTIM_SetWaveform

Function name

__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)

Function description

Set waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (LPTIM_TypeDef * LPTIMx)

Function description

Get actual waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Polarity)

Function description

Set output polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVPOL LL_LPTIM_SetPolarity

LL_LPTIM_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPolarity (LPTIM_TypeDef * LPTIMx)

Function description

Get actual output polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Reference Manual to LL API cross reference:

- CFGR WAVPOL LL_LPTIM_GetPolarity

LL_LPTIM_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)

Function description

Set actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (LPTIM_TypeDef * LPTIMx)

Function description

Get actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 1 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT1_SRC_GPIO
  - LL_LPTIM_INPUT1_SRC_COMP1
  - LL_LPTIM_INPUT1_SRC_COMP3
  - LL_LPTIM_INPUT1_SRC_COMP5 (*)
  - LL_LPTIM_INPUT1_SRC_COMP7 (*) (*) Value not defined for all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- OR IN1 LL_LPTIM_SetInput1Src

LL_LPTIM_SetInput2Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput2Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 2 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT2_SRC_GPIO
  - LL_LPTIM_INPUT2_SRC_COMP2
  - LL_LPTIM_INPUT2_SRC_COMP4
  - LL_LPTIM_INPUT2_SRC_COMP6 (*) (*) Value not defined for all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- OR IN2 LL_LPTIM_SetInput2Src

LL_LPTIM_EnableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Enable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.
- The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Disable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- A trigger event arriving when the timer is already started will be ignored.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the timeout function is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name

__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)

Function description

Start the LPTIM counter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name

__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)

Function description

Configure the external trigger used as a trigger event for the LPTIM.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Source:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2
  - LL_LPTIM_TRIG_SOURCE_COMP3
  - LL_LPTIM_TRIG_SOURCE_COMP4
  - LL_LPTIM_TRIG_SOURCE_COMP5 (*)
  - LL_LPTIM_TRIG_SOURCE_COMP6 (*)
  - LL_LPTIM_TRIG_SOURCE_COMP7 (*)

  (*) Value not defined in all devices.
-
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- An internal clock source must be present when a digital filter is required for the trigger.

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_ConfigTrigger
- CFGR TRGFLT LL_LPTIM_ConfigTrigger
- CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2
  - LL_LPTIM_TRIG_SOURCE_COMP3
  - LL_LPTIM_TRIG_SOURCE_COMP4
  - LL_LPTIM_TRIG_SOURCE_COMP5 (*)
  - LL_LPTIM_TRIG_SOURCE_COMP6 (*)
  - LL_LPTIM_TRIG_SOURCE_COMP7 (*)

  (*) Value not defined in all devices.
-

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8

Reference Manual to LL API cross reference:

- CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name

__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)

Function description

Set the source of the clock used by the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (LPTIM_TypeDef * LPTIMx)

Function description

Get actual LPTIM instance clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name

__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)

Function description

Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockFilter:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.
- An internal clock source must be present when a digital filter is required for external clock.

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_ConfigClock
- CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock digital filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_GetClockFilter

LL_LPTIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)

Function description

Configure the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Get actual encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Enable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1.
- LPTIM instance must be configured in continuous mode prior enabling the encoder mode.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Disable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM operates in encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFLAG_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFLAG_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag (CMPMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMPMCF LL_LPTIM_ClearFLAG_CMPM

LL_LPTIM_IsActiveFlag_CMPM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a compare match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMPM LL_LPTIM_IsActiveFlag_CMPM

LL_LPTIM_ClearFLAG_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFLAG_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload match flag (ARRMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARRMCF LL_LPTIM_ClearFLAG_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a autoreload match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Clear the external trigger valid edge flag(EXTTRIGCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a valid edge on the selected external trigger input has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMPOKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMPOKCF LL_LPTIM_ClearFlag_CMPOK

LL_LPTIM_IsActiveFlag_CMPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CMP register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMPOK LL_LPTIM_IsActiveFlag_CMPOK

LL_LPTIM_ClearFlag_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload register update interrupt flag (ARROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to up interrupt flag (UPCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_EnableIT_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare match interrupt (CMPMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_EnableIT_CMPM

LL_LPTIM_DisableIT_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare match interrupt (CMPMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_DisableIT_CMPM

LL_LPTIM_IsEnabledIT_CMPM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare match interrupt (CMPMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_IsEnabledIT_CMPM

LL_LPTIM_EnableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload match interrupt (ARRMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Enable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Disable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_EnableIT_CMPOK

LL_LPTIM_DisableIT_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_DisableIT_CMPOK

LL_LPTIM_IsEnabledIT_CMPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMPOKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_IsEnabledIT_CMPOK

LL_LPTIM_EnableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

81.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

81.3.1 LPTIM

LPTIM

Input1 Source

LL_LPTIM_INPUT1_SRC_GPIO

LL_LPTIM_INPUT1_SRC_COMP1

LL_LPTIM_INPUT1_SRC_COMP3

LL_LPTIM_INPUT1_SRC_COMP5

LL_LPTIM_INPUT1_SRC_COMP7

Input2 Source

LL_LPTIM_INPUT2_SRC_GPIO

LL_LPTIM_INPUT2_SRC_COMP2

LL_LPTIM_INPUT2_SRC_COMP4

LL_LPTIM_INPUT2_SRC_COMP6

Clock Filter

LL_LPTIM_CLK_FILTER_NONE

Any external clock signal level change is considered as a valid transition

LL_LPTIM_CLK_FILTER_2

External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_4

External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_8

External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING

The rising edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_RISING_FALLING

Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL

LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

LL_LPTIM_CLK_SOURCE_EXTERNAL

LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL

The counter is incremented following each internal clock pulse

LL_LPTIM_COUNTER_MODE_EXTERNAL

The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING

The rising edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_RISING_FALLING

Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMPM

Compare match

LL_LPTIM_ISR_ARRM

Autoreload match

LL_LPTIM_ISR_EXTTRIG

External trigger edge event

LL_LPTIM_ISR_CMPOK

Compare register update OK

LL_LPTIM_ISR_ARROK

Autoreload register update OK

LL_LPTIM_ISR_UP

Counter direction change down to up

LL_LPTIM_ISR_DOWN

Counter direction change up to down

IT Defines

LL_LPTIM_IER_CMPMIE

Compare match Interrupt Enable

LL_LPTIM_IER_ARRMIE

Autoreload match Interrupt Enable

LL_LPTIM_IER_EXTTRIGIE

External trigger valid edge Interrupt Enable

LL_LPTIM_IER_CMPOKIE

Compare register update OK Interrupt Enable

LL_LPTIM_IER_ARROKIE

Autoreload register update OK Interrupt Enable

LL_LPTIM_IER_UPIE

Direction change to UP Interrupt Enable

LL_LPTIM_IER_DOWNIE

Direction change to down Interrupt Enable

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS

LP Timer starts in continuous mode

LL_LPTIM_OPERATING_MODE_ONESHOT

LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR

The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CMP registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE

The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CMP registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM

LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINOUS or SINGLE

LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1

Prescaler division factor is set to 1

LL_LPTIM_PRESCALER_DIV2

Prescaler division factor is set to 2

LL_LPTIM_PRESCALER_DIV4

Prescaler division factor is set to 4

LL_LPTIM_PRESCALER_DIV8

Prescaler division factor is set to 8

LL_LPTIM_PRESCALER_DIV16

Prescaler division factor is set to 16

LL_LPTIM_PRESCALER_DIV32

Prescaler division factor is set to 32

LL_LPTIM_PRESCALER_DIV64

Prescaler division factor is set to 64

LL_LPTIM_PRESCALER_DIV128

Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE

Any trigger active level change is considered as a valid trigger

LL_LPTIM_TRIG_FILTER_2

Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_4

Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_8

Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING

LPTIM counter starts when a rising edge is detected

LL_LPTIM_TRIG_POLARITY_FALLING

LPTIM counter starts when a falling edge is detected

LL_LPTIM_TRIG_POLARITY_RISING_FALLING

LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO

External input trigger is connected to TIMx_ETR input

LL_LPTIM_TRIG_SOURCE_RTCALARMA

External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB

External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1

External input trigger is connected to RTC Tamper 1

LL_LPTIM_TRIG_SOURCE_RTCTAMP2

External input trigger is connected to RTC Tamper 2

LL_LPTIM_TRIG_SOURCE_RTCTAMP3

External input trigger is connected to RTC Tamper 3

LL_LPTIM_TRIG_SOURCE_COMP1

External input trigger is connected to COMP1 output

LL_LPTIM_TRIG_SOURCE_COMP2

External input trigger is connected to COMP2 output

LL_LPTIM_TRIG_SOURCE_COMP3

External input trigger is connected to COMP3 output

LL_LPTIM_TRIG_SOURCE_COMP4

External input trigger is connected to COMP4 output

LL_LPTIM_TRIG_SOURCE_COMP5

External input trigger is connected to COMP5 output

LL_LPTIM_TRIG_SOURCE_COMP6

External input trigger is connected to COMP6 output

LL_LPTIM_TRIG_SOURCE_COMP7

External input trigger is connected to COMP7 output

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE

Preload is disabled: registers are updated after each APB bus write access

LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

82 LL LPUART Generic Driver

82.1 LPUART Firmware driver registers structures

82.1.1 LL_LPUART_InitTypeDef

LL_LPUART_InitTypeDef is defined in the stm32g4xx_ll_lpuart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***

  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***LPUART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***

  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***

  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***

  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

82.2 LPUART Firmware driver API description

The following section lists the various functions of the LPUART library.

82.2.1 Detailed description of functions

LL_LPUART_Enable

Function name

__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)

Function description

LPUART Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name

__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)

Function description

LPUART Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.
- In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name

__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name

__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (USART_TypeDef * LPUARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (USART_TypeDef * LPUARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (USART_TypeDef * LPUARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART enabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART disabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is disabled, LPUART is not able to wake up the MCU from Stop mode

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name

__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_SetTransferDirection
- CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (USART_TypeDef * LPUARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_GetTransferDirection
- CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name

__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_SetParity
- CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetParity (USART_TypeDef * LPUARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_GetParity
- CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance
- **Method:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (USART_TypeDef * LPUARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name

__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)

Function description

Set Word length (nb of data bits, excluding start and stop bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (USART_TypeDef * LPUARTx)

Function description

Return Word length (i.e.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)

Function description

Prevent Mute Mode use.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (USART_TypeDef * LPUARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name

__STATIC_INLINE void LL_LPUART_SetPrescaler (USART_TypeDef * LPUARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (USART_TypeDef * LPUARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (USART_TypeDef * LPUARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name

__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_ConfigCharacter
- CR1 PCE LL_LPUART_ConfigCharacter
- CR1 M LL_LPUART_ConfigCharacter
- CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **LPUARTx:** LPUART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (USART_TypeDef * LPUARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (USART_TypeDef * LPUARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (USART_TypeDef * LPUARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **LPUARTx:** LPUART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (USART_TypeDef * LPUARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

LL_LPUART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (USART_TypeDef * LPUARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the LPUART node.

Parameters

- **LPUARTx:** LPUART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the LPUART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_ConfigNodeAddress
- CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (USART_TypeDef * LPUARTx)

Function description

Return 8 bit Address of the LPUART node as set in ADD field of CR2.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (USART_TypeDef * LPUARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_SetHWFlowCtrl
- CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_GetHWFlowCtrl
- CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL_LPUART_SetWKUPType

Function name

__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Type:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (USART_TypeDef * LPUARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name

__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)

Function description

Configure LPUART BRR register for achieving expected Baud Rate value.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
- Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value != 0).
- Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)

Function description

Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (USART_TypeDef * LPUARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : c

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_SetDEAssertionTime

LL_LPUART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (USART_TypeDef * LPUARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name

__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name

__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (USART_TypeDef * LPUARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_IsEnabledDEMode

LL_LPUART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (USART_TypeDef * LPUARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Framing Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Noise error detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART OverRun Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE line detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS interrupt Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Busy Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Send Break Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from stop mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)

Function description

Clear Parity Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)

Function description

Clear Framing Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)

Function description

Clear Noise detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TXFE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TXFE (USART_TypeDef * LPUARTx)

Function description

Clear TX FIFO Empty Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXFECF LL_LPUART_ClearFlag_TXFE

LL_LPUART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)

Function description

Clear Character Match Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR WUCF LL_LPUART_ClearFlag_WKUP

LL_LPUART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Enable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Enable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Disable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Disable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE Interrupt source is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Check if LPUART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (USART_TypeDef * LPUARTx, uint32_t Direction)

Function description

Get the LPUART data register address used for DMA transfer.

Parameters

- **LPUARTx:** LPUART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_LPUART_DMA_REG_DATA_TRANSMIT
  - LL_LPUART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_DMA_GetRegAddr
-
- TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name

__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name

__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name

__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)

Function description

Request Break sending.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)

Function description

Put LPUART in mute mode and set the RWU flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_DeInit

Function name

ErrorStatus LL_LPUART_DeInit (USART_TypeDef * LPUARTx)

Function description

De-initialize LPUART registers (Registers restored to their default values).

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are de-initialized
  - ERROR: not applicable

LL_LPUART_Init

Function name

ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.

Parameters

- **LPUARTx:** LPUART Instance
- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content
  - ERROR: Problem occurred during LPUART Registers initialization

Notes

- As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name

void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Set each LL_LPUART_InitTypeDef field to default value.

Parameters

- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

82.3 LPUART Firmware driver defines

The following section lists the various define and macros of the module.

82.3.1 LPUART

LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_LPUART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_LPUART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_LPUART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF

Parity error flag

LL_LPUART_ICR_FECF

Framing error flag

LL_LPUART_ICR_NCF

Noise error detected flag

LL_LPUART_ICR_ORECF

Overrun error flag

LL_LPUART_ICR_IDLECF

Idle line detected flag

LL_LPUART_ICR_TXFECF

TX FIFO Empty Clear flag

LL_LPUART_ICR_TCCF

Transmission complete flag

LL_LPUART_ICR_CTSCF

CTS flag

LL_LPUART_ICR_CMCF

Character match flag

LL_LPUART_ICR_WUCF

Wakeup from Stop mode flag

Datawidth

LL_LPUART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_LPUART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_LPUART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH

DE signal is active high

LL_LPUART_DE_POLARITY_LOW

DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_LPUART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_LPUART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_LPUART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_LPUART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_LPUART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_LPUART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_LPUART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_LPUART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_LPUART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE

Parity error flag

LL_LPUART_ISR_FE

Framing error flag

LL_LPUART_ISR_NE

Noise detected flag

LL_LPUART_ISR_ORE

Overrun error flag

LL_LPUART_ISR_IDLE

Idle line detected flag

LL_LPUART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_LPUART_ISR_TC

Transmission complete flag

LL_LPUART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_LPUART_ISR_CTSIF

CTS interrupt flag

LL_LPUART_ISR_CTS

CTS flag

LL_LPUART_ISR_BUSY

Busy flag

LL_LPUART_ISR_CMF

Character match flag

LL_LPUART_ISR_SBKF

Send break flag

LL_LPUART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_LPUART_ISR_WUF

Wakeup from Stop mode flag

LL_LPUART_ISR_TEACK

Transmit enable acknowledge flag

LL_LPUART_ISR_REACK

Receive enable acknowledge flag

LL_LPUART_ISR_TXFE

TX FIFO empty flag

LL_LPUART_ISR_RXFF

RX FIFO full flag

LL_LPUART_ISR_RXFT

RX FIFO threshold flag

LL_LPUART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_LPUART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_LPUART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_LPUART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE

IDLE interrupt enable

LL_LPUART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_LPUART_CR1_TCIE

Transmission complete interrupt enable

LL_LPUART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_LPUART_CR1_PEIE

Parity error

LL_LPUART_CR1_CMIE

Character match interrupt enable

LL_LPUART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_LPUART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_LPUART_CR3_EIE

Error interrupt enable

LL_LPUART_CR3_CTSIE

CTS interrupt enable

LL_LPUART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_LPUART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_LPUART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE

Parity control disabled

LL_LPUART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1

Input clock not devided

LL_LPUART_PRESCALER_DIV2

Input clock devided by 2

LL_LPUART_PRESCALER_DIV4

Input clock devided by 4

LL_LPUART_PRESCALER_DIV6

Input clock devided by 6

LL_LPUART_PRESCALER_DIV8

Input clock devided by 8

LL_LPUART_PRESCALER_DIV10

Input clock devided by 10

LL_LPUART_PRESCALER_DIV12

Input clock devided by 12

LL_LPUART_PRESCALER_DIV16

Input clock devided by 16

LL_LPUART_PRESCALER_DIV32

Input clock devided by 32

LL_LPUART_PRESCALER_DIV64

Input clock devided by 64

LL_LPUART_PRESCALER_DIV128

Input clock devided by 128

LL_LPUART_PRESCALER_DIV256

Input clock devided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_LPUART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1

1 stop bit

LL_LPUART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_LPUART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_LPUART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE

LPUART wake up from Mute mode on Idle Line

LL_LPUART_WAKEUP_ADDRESSMARK

LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_LPUART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_LPUART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

83 LL OPAMP Generic Driver

83.1 OPAMP Firmware driver registers structures

83.1.1 LL_OPAMP_InitTypeDef

LL_OPAMP_InitTypeDef is defined in the stm32g4xx_ll_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t FunctionalMode***
- ***uint32_t InputNonInverting***
- ***uint32_t InputInverting***

Field Documentation

- ***uint32_t LL_OPAMP_InitTypeDef::PowerMode***

  Set OPAMP power mode. This parameter can be a value of OPAMP_LL_EC_POWERMODEThis feature can be modified afterwards using unitary function **LL_OPAMP_SetPowerMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::FunctionalMode***

  Set OPAMP functional mode by setting internal connections: OPAMP operation in standalone, follower, ... This parameter can be a value of ***OPAMP_LL_EC_FUNCTIONAL_MODE***

  Note:

  – If OPAMP is configured in mode PGA, the gain can be configured using function **LL_OPAMP_SetPGAGain()**.

  This feature can be modified afterwards using unitary function **LL_OPAMP_SetFunctionalMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputNonInverting***

  Set OPAMP input non-inverting connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_NONINVERTING***This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputNonInverting()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputInverting***

  Set OPAMP inverting input connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_INVERTING***

  Note:

  – OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin), this parameter is discarded.

  This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputInverting()**.

83.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

83.2.1 Detailed description of functions

LL_OPAMP_SetMode

Function name

__STATIC_INLINE void LL_OPAMP_SetMode (OPAMP_TypeDef * OPAMPx, uint32_t Mode)

Function description

Set OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance
- **Mode:** This parameter can be one of the following values:
  - LL_OPAMP_MODE_FUNCTIONAL
  - LL_OPAMP_MODE_CALIBRATION

Return values

- **None:**

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_SetMode

LL_OPAMP_GetMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_MODE_FUNCTIONAL
  - LL_OPAMP_MODE_CALIBRATION

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_GetMode

LL_OPAMP_SetFunctionalMode

Function name

__STATIC_INLINE void LL_OPAMP_SetFunctionalMode (OPAMP_TypeDef * OPAMPx, uint32_t FunctionalMode)

Function description

Set OPAMP functional mode by setting internal connections.

Parameters

- **OPAMPx:** OPAMP instance
- **FunctionalMode:** This parameter can be one of the following values:
  - LL_OPAMP_MODE_STANDALONE
  - LL_OPAMP_MODE_FOLLOWER
  - LL_OPAMP_MODE_PGA
  - LL_OPAMP_MODE_PGA_IO0
  - LL_OPAMP_MODE_PGA_IO0_BIAS
  - LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

Return values

- **None:**

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetFunctionalMode

LL_OPAMP_GetFunctionalMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetFunctionalMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP functional mode from setting of internal connections.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_MODE_STANDALONE
  - LL_OPAMP_MODE_FOLLOWER
  - LL_OPAMP_MODE_PGA
  - LL_OPAMP_MODE_PGA_IO0
  - LL_OPAMP_MODE_PGA_IO0_BIAS
  - LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetFunctionalMode

LL_OPAMP_SetPGAGain

Function name

__STATIC_INLINE void LL_OPAMP_SetPGAGain (OPAMP_TypeDef * OPAMPx, uint32_t PGAGain)

Function description

Set OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance
- **PGAGain:** This parameter can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2_OR_MINUS_1
  - LL_OPAMP_PGA_GAIN_4_OR_MINUS_3
  - LL_OPAMP_PGA_GAIN_8_OR_MINUS_7
  - LL_OPAMP_PGA_GAIN_16_OR_MINUS_15
  - LL_OPAMP_PGA_GAIN_32_OR_MINUS_31
  - LL_OPAMP_PGA_GAIN_64_OR_MINUS_63

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_SetPGAGain

LL_OPAMP_GetPGAGain

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPGAGain (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2_OR_MINUS_1
  - LL_OPAMP_PGA_GAIN_4_OR_MINUS_3
  - LL_OPAMP_PGA_GAIN_8_OR_MINUS_7
  - LL_OPAMP_PGA_GAIN_16_OR_MINUS_15
  - LL_OPAMP_PGA_GAIN_32_OR_MINUS_31
  - LL_OPAMP_PGA_GAIN_64_OR_MINUS_63

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_GetPGAGain

LL_OPAMP_SetPowerMode

Function name

__STATIC_INLINE void LL_OPAMP_SetPowerMode (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode)

Function description

Set OPAMP power mode normal or highspeed.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED

Return values

- **None:**

Notes

- OPAMP highspeed mode allows output stage to have a better slew rate.

Reference Manual to LL API cross reference:

- CSR HIGHSPEEDEN LL_OPAMP_SetPowerMode

LL_OPAMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPowerMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP power mode normal or highspeed.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED

Notes

- OPAMP highspeed mode allows output stage to have a better slew rate.

Reference Manual to LL API cross reference:

- CSR HIGHSPEEDEN LL_OPAMP_GetPowerMode

LL_OPAMP_SetInputNonInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputNonInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputNonInverting)

Function description

Set OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputNonInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINVERT_IO1
  - LL_OPAMP_INPUT_NONINVERT_IO2
  - LL_OPAMP_INPUT_NONINVERT_IO3
  - LL_OPAMP_INPUT_NONINVERT_DAC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_SetInputNonInverting

LL_OPAMP_GetInputNonInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputNonInverting (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINVERT_IO1
  - LL_OPAMP_INPUT_NONINVERT_IO2
  - LL_OPAMP_INPUT_NONINVERT_IO3
  - LL_OPAMP_INPUT_NONINVERT_DAC

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_GetInputNonInverting

LL_OPAMP_SetInputInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputInverting)

Function description

Set OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0
  - LL_OPAMP_INPUT_INVERT_IO1
  - LL_OPAMP_INPUT_INVERT_CONNECT_NO

Return values

- **None:**

Notes

- OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetInputInverting

LL_OPAMP_GetInputInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputInverting (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0
  - LL_OPAMP_INPUT_INVERT_IO1
  - LL_OPAMP_INPUT_INVERT_CONNECT_NO

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetInputInverting

LL_OPAMP_SetInputNonInvertingSecondary

Function name

__STATIC_INLINE void LL_OPAMP_SetInputNonInvertingSecondary (OPAMP_TypeDef * OPAMPx, uint32_t InputNonInverting)

Function description

Set OPAMP non-inverting input secondary connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputNonInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0_SEC
  - LL_OPAMP_INPUT_NONINVERT_IO1_SEC
  - LL_OPAMP_INPUT_NONINVERT_IO2_SEC
  - LL_OPAMP_INPUT_NONINVERT_IO3_SEC
  - LL_OPAMP_INPUT_NONINVERT_DAC_SEC

Return values

- **None:**

Reference Manual to LL API cross reference:

- TCMR VPSSEL LL_OPAMP_SetInputNonInvertingSecondary

LL_OPAMP_GetInputNonInvertingSecondary

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputNonInvertingSecondary (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP non-inverting input secondary connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0_SEC
  - LL_OPAMP_INPUT_NONINVERT_IO1_SEC
  - LL_OPAMP_INPUT_NONINVERT_IO2_SEC
  - LL_OPAMP_INPUT_NONINVERT_IO3_SEC
  - LL_OPAMP_INPUT_NONINVERT_DAC_SEC

Reference Manual to LL API cross reference:

- TCMR VPSSEL LL_OPAMP_GetInputNonInvertingSecondary

LL_OPAMP_SetInputInvertingSecondary

Function name

__STATIC_INLINE void LL_OPAMP_SetInputInvertingSecondary (OPAMP_TypeDef * OPAMPx, uint32_t InputInverting)

Function description

Set OPAMP inverting input secondary connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0_SEC
  - LL_OPAMP_INPUT_INVERT_IO1_SEC
  - LL_OPAMP_INPUT_INVERT_PGA_SEC
  - LL_OPAMP_INPUT_INVERT_FOLLOWER_SEC

Return values

- **None:**

Notes

- OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

Reference Manual to LL API cross reference:

- TCMR VMSSEL LL_OPAMP_SetInputInvertingSecondary

LL_OPAMP_GetInputInvertingSecondary

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputInvertingSecondary (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP inverting input secondary connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0_SEC
  - LL_OPAMP_INPUT_INVERT_IO1_SEC
  - LL_OPAMP_INPUT_INVERT_PGA_SEC
  - LL_OPAMP_INPUT_INVERT_FOLLOWER_SEC

Reference Manual to LL API cross reference:

- TCMR VMSSEL LL_OPAMP_GetInputInvertingSecondary

LL_OPAMP_SetInputsMuxMode

Function name

__STATIC_INLINE void LL_OPAMP_SetInputsMuxMode (OPAMP_TypeDef * OPAMPx, uint32_t InputsMuxMode)

Function description

Set OPAMP inputs multiplexer mode.

Parameters

- **OPAMPx:** OPAMP instance
- **InputsMuxMode:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_MUX_DISABLE
  - LL_OPAMP_INPUT_MUX_TIM1_CH6
  - LL_OPAMP_INPUT_MUX_TIM8_CH6
  - LL_OPAMP_INPUT_MUX_TIM20_CH6 (1) On this STM32 serie, this value is not available on all devices. Refer to datasheet for details.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TCMR TCMEN LL_OPAMP_SetInputsMuxMode

LL_OPAMP_GetInputsMuxMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputsMuxMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP inputs multiplexer mode.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_MUX_DISABLE
  - LL_OPAMP_INPUT_MUX_TIM1_CH6
  - LL_OPAMP_INPUT_MUX_TIM8_CH6
  - LL_OPAMP_INPUT_MUX_TIM20_CH6 (1) On this STM32 serie, this value is not available on all devices. Refer to datasheet for details.

Reference Manual to LL API cross reference:

- TCMR TCMEN LL_OPAMP_GetInputsMuxMode

LL_OPAMP_SetInternalOutput

Function name

__STATIC_INLINE void LL_OPAMP_SetInternalOutput (OPAMP_TypeDef * OPAMPx, uint32_t InternalOutput)

Function description

Set OPAMP internal output.

Parameters

- **OPAMPx:** OPAMP instance
- **InternalOutput:** This parameter can be one of the following values:
  - LL_OPAMP_INTERNAL_OUPUT_DISABLED
  - LL_OPAMP_INTERNAL_OUPUT_ENABLED

Return values

- **None:**

Notes

- OPAMP internal output is used to link OPAMP output to ADC input internally.

Reference Manual to LL API cross reference:

- CSR OPAMPINTEN LL_OPAMP_SetInternalOutput

LL_OPAMP_GetInternalOutput

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInternalOutput (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP internal output state.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INTERNAL_OUPUT_DISABLED
  - LL_OPAMP_INTERNAL_OUPUT_ENABLED

Reference Manual to LL API cross reference:

- CSR OPAMPINTEN LL_OPAMP_GetInternalOutput

LL_OPAMP_SetTrimmingMode

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingMode (OPAMP_TypeDef * OPAMPx, uint32_t TrimmingMode)

Function description

Set OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance
- **TrimmingMode:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_FACTORY
  - LL_OPAMP_TRIMMING_USER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_SetTrimmingMode

LL_OPAMP_GetTrimmingMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_FACTORY
  - LL_OPAMP_TRIMMING_USER

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_GetTrimmingMode

LL_OPAMP_SetCalibrationSelection

Function name

__STATIC_INLINE void LL_OPAMP_SetCalibrationSelection (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair)

Function description

Set OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS (1)
  - LL_OPAMP_TRIMMING_PMOS (1)
  - LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA
  - LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

  (1) Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS)

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_SetCalibrationSelection

LL_OPAMP_GetCalibrationSelection

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetCalibrationSelection (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS (1)
  - LL_OPAMP_TRIMMING_PMOS (1)
  - LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA
  - LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

  (1) Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS)

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_GetCalibrationSelection

LL_OPAMP_IsCalibrationOutputSet

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsCalibrationOutputSet (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP calibration result of toggling output.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Notes

- This functions returns: 0 if OPAMP calibration output is reset 1 if OPAMP calibration output is set

Reference Manual to LL API cross reference:

- CSR OUTCAL LL_OPAMP_IsCalibrationOutputSet

LL_OPAMP_SetTrimmingValue

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair, uint32_t TrimmingValue)

Function description

Set OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS
- **TrimmingValue:** 0x00...0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR TRIMOFFSETN LL_OPAMP_SetTrimmingValue
- CSR TRIMOFFSETP LL_OPAMP_SetTrimmingValue

LL_OPAMP_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair)

Function description

Get OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS

Return values

- **0x0...0x1F:**

Reference Manual to LL API cross reference:

- CSR TRIMOFFSETN LL_OPAMP_GetTrimmingValue
- CSR TRIMOFFSETP LL_OPAMP_GetTrimmingValue

LL_OPAMP_Enable

Function name

__STATIC_INLINE void LL_OPAMP_Enable (OPAMP_TypeDef * OPAMPx)

Function description

Enable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Notes

- After enable from off state, OPAMP requires a delay to fullfill wake up time specification. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Enable

LL_OPAMP_Disable

Function name

__STATIC_INLINE void LL_OPAMP_Disable (OPAMP_TypeDef * OPAMPx)

Function description

Disable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Disable

LL_OPAMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsEnabled (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP instance enable state (0: OPAMP is disabled, 1: OPAMP is enabled)

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_IsEnabled

LL_OPAMP_Lock

Function name

__STATIC_INLINE void LL_OPAMP_Lock (OPAMP_TypeDef * OPAMPx)

Function description

Lock OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Notes

- Once locked, OPAMP configuration can be accessed in read-only.
- The only way to unlock the OPAMP is a device hardware reset.

Reference Manual to LL API cross reference:

- CSR LOCK LL_OPAMP_Lock

LL_OPAMP_IsLocked

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsLocked (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP lock state (0: OPAMP is unlocked, 1: OPAMP is locked).

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Notes

- Once locked, OPAMP configuration can be accessed in read-only.
- The only way to unlock the OPAMP is a device hardware reset.

Reference Manual to LL API cross reference:

- CSR LOCK LL_OPAMP_IsLocked

LL_OPAMP_LockTimerMux

Function name

__STATIC_INLINE void LL_OPAMP_LockTimerMux (OPAMP_TypeDef * OPAMPx)

Function description

Lock OPAMP instance timer controlled mux.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Notes

- Once locked, OPAMP timer controlled mux configuration can be accessed in read-only.
- The only way to unlock the OPAMP timer controlled mux is a device hardware reset.

Reference Manual to LL API cross reference:

- TCMR LOCK LL_OPAMP_LockTimerMux

LL_OPAMP_IsTimerMuxLocked

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsTimerMuxLocked (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP timer controlled mux lock state (0: OPAMP timer controlled mux is unlocked, 1: OPAMP timer controlled mux is locked).

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Notes

- Once locked, OPAMP timer controlled mux configuration can be accessed in read-only.
- The only way to unlock the OPAMP timer controlled mux is a device hardware reset.

Reference Manual to LL API cross reference:

- TCMR LOCK LL_OPAMP_IsTimerMuxLocked

LL_OPAMP_DeInit

Function name

ErrorStatus LL_OPAMP_DeInit (OPAMP_TypeDef * OPAMPx)

Function description

De-initialize registers of the selected OPAMP instance to their default reset values.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: OPAMP registers are de-initialized
  - ERROR: OPAMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_OPAMP_Init

Function name

ErrorStatus LL_OPAMP_Init (OPAMP_TypeDef * OPAMPx, LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Initialize some features of OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance
- **OPAMP_InitStruct:** Pointer to a LL_OPAMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: OPAMP registers are initialized
  - ERROR: OPAMP registers are not initialized

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

LL_OPAMP_StructInit

Function name

void LL_OPAMP_StructInit (LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Set each LL_OPAMP_InitTypeDef field to default value.

Parameters

- **OPAMP_InitStruct:** pointer to a LL_OPAMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

83.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

83.3.1 OPAMP

OPAMP

OPAMP functional mode

LL_OPAMP_MODE_STANDALONE

OPAMP functional mode, OPAMP operation in standalone

LL_OPAMP_MODE_FOLLOWER

OPAMP functional mode, OPAMP operation in follower

LL_OPAMP_MODE_PGA

OPAMP functional mode, OPAMP operation in PGA

LL_OPAMP_MODE_PGA_IO0

In PGA mode, the inverting input is connected to VINM0 for filtering

LL_OPAMP_MODE_PGA_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0

LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0

Definitions of OPAMP hardware constraints delays

LL_OPAMP_DELAY_STARTUP_US

Delay for OPAMP startup time

OPAMP input inverting

LL_OPAMP_INPUT_INVERT_IO0

OPAMP inverting input connected to I/O VINM0 (PA3 for OPAMP1, PA5 for OPAMP2, PB2 for OPAMP3, PB10 for OPAMP4, PB15 for OPAMP5, PA1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_INVERT_IO1

OPAMP inverting input connected to I/0 VINM1 (PC5 for OPAMP1, PC5 for OPAMP2, PB10 for OPAMP3, PB8 for OPAMP4, PA3 for OPAMP5, PB1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_INVERT_CONNECT_NO

OPAMP inverting input not externally connected (intended for OPAMP in mode follower or PGA with positive gain without bias). Note: On this STM32 serie, this literal include cases of value 0x11 for mode follower and value 0x10 for mode PGA.

OPAMP input inverting secondary

LL_OPAMP_INPUT_INVERT_IO0_SEC

OPAMP secondary mode is standalone mode - Only applicable if

LL_OPAMP_INPUT_INVERT_IO1_SEC

OPAMP secondary mode is standalone mode - Only applicable if

LL_OPAMP_INPUT_INVERT_PGA_SEC

OPAMP secondary mode is PGA mode - Only applicable if configured mode through call to

LL_OPAMP_INPUT_INVERT_FOLLOWER_SEC

OPAMP secondary mode is Follower mode - Only applicable if configured mode through call to

OPAMP inputs multiplexer mode

LL_OPAMP_INPUT_MUX_DISABLE

OPAMP inputs timer controlled multiplexer mode disabled.

LL_OPAMP_INPUT_MUX_TIM1_CH6

OPAMP inputs timer controlled multiplexer mode enabled, controlled by TIM1 OC6.

LL_OPAMP_INPUT_MUX_TIM8_CH6

OPAMP inputs timer controlled multiplexer mode enabled, controlled by TIM8 OC6.

LL_OPAMP_INPUT_MUX_TIM20_CH6

OPAMP inputs timer controlled multiplexer mode enabled, controlled by TIM20 OC6. Note: On this STM32 serie, TIM20 is not available on all devices. Refer to device datasheet for more details

OPAMP input non-inverting

LL_OPAMP_INPUT_NONINVERT_IO0

OPAMP non inverting input connected to I/O VINP0 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4, PB14 for OPAMP5, PB12 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO1

OPAMP non inverting input connected to I/O VINP1 (PA3 for OPAMP1, PB14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4, PD12 for OPAMP5, PD9 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO2

OPAMP non inverting input connected to I/O VINP2 (PA7 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PB11 for OPAMP4, PC3 for OPAMP5, PB13 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO3

OPAMP non inverting input connected to I/O VINP3 (PD14 for OPAMP2)

LL_OPAMP_INPUT_NONINVERT_DAC

OPAMP non inverting input connected internally to DAC channel (DAC3_CH1 for OPAMP1, DAC3_CH2 for OPAMP3, DAC4_CH1 for OPAMP4, DAC4_CH2 for OPAMP5, DAC3_CH1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP input non-inverting secondary

LL_OPAMP_INPUT_NONINVERT_IO0_SEC

OPAMP secondary non inverting input connected to I/O VINP0 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4, PB14 for OPAMP5, PB12 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO1_SEC

OPAMP secondary non inverting input connected to I/O VINP1 (PA3 for OPAMP1, PB14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4, PD12 for OPAMP5, PD9 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO2_SEC

OPAMP secondary non inverting input connected to I/O VINP2 (PA7 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PB11 for OPAMP4, PC3 for OPAMP5, PB13 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO3_SEC

OPAMP secondary non inverting input connected to I/O VINP3 (PD14 for OPAMP2)

LL_OPAMP_INPUT_NONINVERT_DAC_SEC

OPAMP secondary non inverting input connected internally to DAC channel (DAC3_CH1 for OPAMP1, DAC3_CH2 for OPAMP3, DAC4_CH1 for OPAMP4, DAC4_CH2 for OPAMP5, DAC3_CH1 for OPAMP6) Note: On this STM32 serie, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP internal output mode

LL_OPAMP_INTERNAL_OUPUT_DISABLED

OPAMP internal output to ADC disabled.

LL_OPAMP_INTERNAL_OUPUT_ENABLED

OPAMP internal output to ADC enabled.

OPAMP mode calibration or functional.

LL_OPAMP_MODE_FUNCTIONAL

OPAMP functional mode

LL_OPAMP_MODE_CALIBRATION

OPAMP calibration mode

OPAMP PGA gain (relevant when OPAMP is in functional mode PGA)

LL_OPAMP_PGA_GAIN_2_OR_MINUS_1

OPAMP PGA gain 2 or -1

LL_OPAMP_PGA_GAIN_4_OR_MINUS_3

OPAMP PGA gain 4 or -3

LL_OPAMP_PGA_GAIN_8_OR_MINUS_7

OPAMP PGA gain 8 or -7

LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

OPAMP PGA gain 16 or -15

LL_OPAMP_PGA_GAIN_32_OR_MINUS_31

OPAMP PGA gain 32 or -31

LL_OPAMP_PGA_GAIN_64_OR_MINUS_63

OPAMP PGA gain 64 or -63

OPAMP PowerMode

LL_OPAMP_POWERMODE_NORMAL

OPAMP output in normal mode

LL_OPAMP_POWERMODE_HIGHSPEED

OPAMP output in highspeed mode

OPAMP trimming mode

LL_OPAMP_TRIMMING_FACTORY

OPAMP trimming factors set to factory values

LL_OPAMP_TRIMMING_USER

OPAMP trimming factors set to user values

OPAMP trimming of transistors differential pair NMOS or PMOS

LL_OPAMP_TRIMMING_NMOS_VREF_90PC_VDDA

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.9*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.5*Vdda).

LL_OPAMP_TRIMMING_PMOS_VREF_10PC_VDDA

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.1*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.33*Vdda).

LL_OPAMP_TRIMMING_NMOS

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.9*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_PMOS

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.1*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

Common write and read registers macro

LL_OPAMP_WriteReg

Description:

- Write a value in OPAMP register.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_OPAMP_ReadReg

Description:

- Read a value in OPAMP register.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be read

Return value:

- Register: value

84 LL PWR Generic Driver

84.1 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

84.1.1 Detailed description of functions

LL_PWR_EnableLowPowerRunMode

Function name

__STATIC_INLINE void LL_PWR_EnableLowPowerRunMode (void)

Function description

Switch the regulator from main mode to low-power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPR LL_PWR_EnableLowPowerRunMode

LL_PWR_DisableLowPowerRunMode

Function name

__STATIC_INLINE void LL_PWR_DisableLowPowerRunMode (void)

Function description

Switch the regulator from low-power mode to main mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPR LL_PWR_DisableLowPowerRunMode

LL_PWR_IsEnabledLowPowerRunMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledLowPowerRunMode (void)

Function description

Check if the regulator is in low-power mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 LPR LL_PWR_IsEnabledLowPowerRunMode

LL_PWR_EnterLowPowerRunMode

Function name

__STATIC_INLINE void LL_PWR_EnterLowPowerRunMode (void)

Function description

Switch from run main mode to run low-power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPR LL_PWR_EnterLowPowerRunMode

LL_PWR_ExitLowPowerRunMode

Function name

__STATIC_INLINE void LL_PWR_ExitLowPowerRunMode (void)

Function description

Switch from run main mode to low-power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPR LL_PWR_ExitLowPowerRunMode

LL_PWR_SetRegulVoltageScaling

Function name

__STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)

Function description

Set the main internal regulator output voltage.

Parameters

- **VoltageScaling:** This parameter can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 VOS LL_PWR_SetRegulVoltageScaling

LL_PWR_GetRegulVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling (void)

Function description

Get the main internal regulator output voltage.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2

Reference Manual to LL API cross reference:

- CR1 VOS LL_PWR_GetRegulVoltageScaling

LL_PWR_EnableRange1BoostMode

Function name

__STATIC_INLINE void LL_PWR_EnableRange1BoostMode (void)

Function description

Enable main regulator voltage range 1 boost mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 R1MODE LL_PWR_EnableRange1BoostMode

LL_PWR_DisableRange1BoostMode

Function name

__STATIC_INLINE void LL_PWR_DisableRange1BoostMode (void)

Function description

Disable main regulator voltage range 1 boost mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 R1MODE LL_PWR_DisableRange1BoostMode

LL_PWR_IsEnabledRange1BoostMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledRange1BoostMode (void)

Function description

Check if the main regulator voltage range 1 boost mode is enabled.

Return values

- **Inverted:** state of bit (0 or 1).

Reference Manual to LL API cross reference:

- CR5 R1MODE LL_PWR_IsEnabledRange1BoostMode

LL_PWR_EnableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DBP LL_PWR_DisableBkUpAccess

LL_PWR_IsEnabledBkUpAccess

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)

Function description

Check if the backup domain is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 DBP LL_PWR_IsEnabledBkUpAccess

LL_PWR_SetPowerMode

Function name

__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t LowPowerMode)

Function description

Set Low-Power mode.

Parameters

- **LowPowerMode:** This parameter can be one of the following values:
  - LL_PWR_MODE_STOP0
  - LL_PWR_MODE_STOP1
  - LL_PWR_MODE_STANDBY
  - LL_PWR_MODE_SHUTDOWN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPMS LL_PWR_SetPowerMode

LL_PWR_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)

Function description

Get Low-Power mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_MODE_STOP0
  - LL_PWR_MODE_STOP1
  - LL_PWR_MODE_STANDBY
  - LL_PWR_MODE_SHUTDOWN

Reference Manual to LL API cross reference:

- CR1 LPMS LL_PWR_GetPowerMode

LL_PWR_EnableUCPDStandbyMode

Function name

__STATIC_INLINE void LL_PWR_EnableUCPDStandbyMode (void)

Function description

Enable the USB Type-C and Power Delivery memorization in Standby mode.

Return values

- **None:**

Notes

- This function must be called just before entering Standby mode.

Reference Manual to LL API cross reference:

- CR3 UCPD_STDBY LL_PWR_EnableUCPDStandbyMode

LL_PWR_DisableUCPDStandbyMode

Function name

__STATIC_INLINE void LL_PWR_DisableUCPDStandbyMode (void)

Function description

Disable the USB Type-C and Power Delivery memorization in Standby mode.

Return values

- **None:**

Notes

- This function must be called after exiting Standby mode and before any UCPD configuration update.

Reference Manual to LL API cross reference:

- CR3 UCPD_STDBY LL_PWR_DisableUCPDStandbyMode

LL_PWR_IsEnabledUCPDStandbyMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUCPDStandbyMode (void)

Function description

Check the USB Type-C and Power Delivery Standby mode memorization state.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 UCPD_STDBY LL_PWR_IsEnabledUCPDStandbyMode

LL_PWR_EnableUCPDDeadBattery

Function name

__STATIC_INLINE void LL_PWR_EnableUCPDDeadBattery (void)

Function description

Enable the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- CR3 UCPD_DBDIS LL_PWR_EnableUCPDDeadBattery

LL_PWR_DisableUCPDDeadBattery

Function name

__STATIC_INLINE void LL_PWR_DisableUCPDDeadBattery (void)

Function description

Disable the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- CR3 UCPD_DBDIS LL_PWR_DisableUCPDDeadBattery

LL_PWR_IsEnabledUCPDDeadBattery

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUCPDDeadBattery (void)

Function description

Check the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **State:** of feature (1 : enabled; 0 : disabled).

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- CR3 UCPD_DBDIS LL_PWR_IsEnabledUCPDDeadBattery

LL_PWR_EnablePVM

Function name

__STATIC_INLINE void LL_PWR_EnablePVM (uint32_t PeriphVoltage)

Function description

Enable the Power Voltage Monitoring on a peripheral.

Parameters

- **PeriphVoltage:** This parameter can be one of the following values:
  - LL_PWR_PVM_VDDA_COMP (*)
  - LL_PWR_PVM_VDDA_FASTDAC (*)
  - LL_PWR_PVM_VDDA_ADC
  - LL_PWR_PVM_VDDA_OPAMP_DAC

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVME1 LL_PWR_EnablePVM
- CR2 PVME2 LL_PWR_EnablePVM
- CR2 PVME3 LL_PWR_EnablePVM
- CR2 PVME4 LL_PWR_EnablePVM

LL_PWR_DisablePVM

Function name

__STATIC_INLINE void LL_PWR_DisablePVM (uint32_t PeriphVoltage)

Function description

Disable the Power Voltage Monitoring on a peripheral.

Parameters

- **PeriphVoltage:** This parameter can be one of the following values:
  - LL_PWR_PVM_VDDA_COMP (*)
  - LL_PWR_PVM_VDDA_FASTDAC (*)
  - LL_PWR_PVM_VDDA_ADC
  - LL_PWR_PVM_VDDA_OPAMP_DAC

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVME1 LL_PWR_DisablePVM
- CR2 PVME2 LL_PWR_DisablePVM
- CR2 PVME3 LL_PWR_DisablePVM
- CR2 PVME4 LL_PWR_DisablePVM

LL_PWR_IsEnabledPVM

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVM (uint32_t PeriphVoltage)

Function description

Check if Power Voltage Monitoring is enabled on a peripheral.

Parameters

- **PeriphVoltage:** This parameter can be one of the following values:
  - LL_PWR_PVM_VDDA_COMP (*)
  - LL_PWR_PVM_VDDA_FASTDAC (*)
  - LL_PWR_PVM_VDDA_ADC
  - LL_PWR_PVM_VDDA_OPAMP_DAC

  (*) value not defined in all devices

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 PVME1 LL_PWR_IsEnabledPVM
- CR2 PVME2 LL_PWR_IsEnabledPVM
- CR2 PVME3 LL_PWR_IsEnabledPVM
- CR2 PVME4 LL_PWR_IsEnabledPVM

LL_PWR_SetPVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)

Function description

Configure the voltage threshold detected by the Power Voltage Detector.

Parameters

- **PVDLevel:** This parameter can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)

Function description

Get the voltage threshold detection.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:

- CR2 PLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name

__STATIC_INLINE void LL_PWR_EnablePVD (void)

Function description

Enable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVDE LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name

__STATIC_INLINE void LL_PWR_DisablePVD (void)

Function description

Disable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVDE LL_PWR_DisablePVD

LL_PWR_IsEnabledPVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)

Function description

Check if Power Voltage Detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 PVDE LL_PWR_IsEnabledPVD

LL_PWR_EnableInternWU

Function name

__STATIC_INLINE void LL_PWR_EnableInternWU (void)

Function description

Enable Internal Wake-up line.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWF LL_PWR_EnableInternWU

LL_PWR_DisableInternWU

Function name

__STATIC_INLINE void LL_PWR_DisableInternWU (void)

Function description

Disable Internal Wake-up line.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWF LL_PWR_DisableInternWU

LL_PWR_IsEnabledInternWU

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledInternWU (void)

Function description

Check if Internal Wake-up line is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIWF LL_PWR_IsEnabledInternWU

LL_PWR_EnablePUPDCfg

Function name

__STATIC_INLINE void LL_PWR_EnablePUPDCfg (void)

Function description

Enable pull-up and pull-down configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 APC LL_PWR_EnablePUPDCfg

LL_PWR_DisablePUPDCfg

Function name

__STATIC_INLINE void LL_PWR_DisablePUPDCfg (void)

Function description

Disable pull-up and pull-down configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 APC LL_PWR_DisablePUPDCfg

LL_PWR_IsEnabledPUPDCfg

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPUPDCfg (void)

Function description

Check if pull-up and pull-down configuration is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 APC LL_PWR_IsEnabledPUPDCfg

LL_PWR_EnableSRAM2Retention

Function name

__STATIC_INLINE void LL_PWR_EnableSRAM2Retention (void)

Function description

Enable SRAM2 content retention in Standby mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RRS LL_PWR_EnableSRAM2Retention

LL_PWR_DisableSRAM2Retention

Function name

__STATIC_INLINE void LL_PWR_DisableSRAM2Retention (void)

Function description

Disable SRAM2 content retention in Standby mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RRS LL_PWR_DisableSRAM2Retention

LL_PWR_IsEnabledSRAM2Retention

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledSRAM2Retention (void)

Function description

Check if SRAM2 content retention in Standby mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 RRS LL_PWR_IsEnabledSRAM2Retention

LL_PWR_EnableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWUP1 LL_PWR_EnableWakeUpPin
- CR3 EWUP2 LL_PWR_EnableWakeUpPin
- CR3 EWUP3 LL_PWR_EnableWakeUpPin
- CR3 EWUP4 LL_PWR_EnableWakeUpPin
- CR3 EWUP5 LL_PWR_EnableWakeUpPin
-

LL_PWR_DisableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWUP1 LL_PWR_DisableWakeUpPin
- CR3 EWUP2 LL_PWR_DisableWakeUpPin
- CR3 EWUP3 LL_PWR_DisableWakeUpPin
- CR3 EWUP4 LL_PWR_DisableWakeUpPin
- CR3 EWUP5 LL_PWR_DisableWakeUpPin
-

LL_PWR_IsEnabledWakeUpPin

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)

Function description

Check if the WakeUp PINx functionality is enabled.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EWUP1 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP2 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP3 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP4 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP5 LL_PWR_IsEnabledWakeUpPin
-

LL_PWR_SetBattChargResistor

Function name

__STATIC_INLINE void LL_PWR_SetBattChargResistor (uint32_t Resistor)

Function description

Set the resistor impedance.

Parameters

- **Resistor:** This parameter can be one of the following values:
  - LL_PWR_BATT_CHARG_RESISTOR_5K
  - LL_PWR_BATT_CHARGRESISTOR_1_5K

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 VBRS LL_PWR_SetBattChargResistor

LL_PWR_GetBattChargResistor

Function name

__STATIC_INLINE uint32_t LL_PWR_GetBattChargResistor (void)

Function description

Get the resistor impedance.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_BATT_CHARG_RESISTOR_5K
  - LL_PWR_BATT_CHARGRESISTOR_1_5K

Reference Manual to LL API cross reference:

- CR4 VBRS LL_PWR_GetBattChargResistor

LL_PWR_EnableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_EnableBatteryCharging (void)

Function description

Enable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 VBE LL_PWR_EnableBatteryCharging

LL_PWR_DisableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_DisableBatteryCharging (void)

Function description

Disable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 VBE LL_PWR_DisableBatteryCharging

LL_PWR_IsEnabledBatteryCharging

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBatteryCharging (void)

Function description

Check if battery charging is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR4 VBE LL_PWR_IsEnabledBatteryCharging

LL_PWR_SetWakeUpPinPolarityLow

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin polarity low for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 WP1 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP2 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP3 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP4 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP5 LL_PWR_SetWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPolarityHigh

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityHigh (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin polarity high for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 WP1 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP2 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP3 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP4 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP5 LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_IsWakeUpPinPolarityLow

Function name

__STATIC_INLINE uint32_t LL_PWR_IsWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Get the Wake-Up pin polarity for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR4 WP1 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP2 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP3 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP4 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP5 LL_PWR_IsWakeUpPinPolarityLow

LL_PWR_EnableGPIOPullUp

Function name

__STATIC_INLINE void LL_PWR_EnableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F
  - LL_PWR_GPIO_G

  (*) value not defined in all devices
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- PUCRA PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRB PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRC PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRD PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRE PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRF PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRG PU0-15 LL_PWR_EnableGPIOPullUp
-

LL_PWR_DisableGPIOPullUp

Function name

__STATIC_INLINE void LL_PWR_DisableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F
  - LL_PWR_GPIO_G

  (*) value not defined in all devices
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- PUCRA PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRB PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRC PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRD PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRE PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRF PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRG PU0-15 LL_PWR_DisableGPIOPullUp
-

LL_PWR_IsEnabledGPIOPullUp

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Check if GPIO pull-up state is enabled.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F
  - LL_PWR_GPIO_G

  (*) value not defined in all devices
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PUCRA PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRB PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRC PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRD PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRE PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRF PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRG PU0-15 LL_PWR_IsEnabledGPIOPullUp
-

LL_PWR_EnableGPIOPullDown

Function name

__STATIC_INLINE void LL_PWR_EnableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F
  - LL_PWR_GPIO_G

  (*) value not defined in all devices
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- PDCRA PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRB PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRC PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRD PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRE PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRF PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRG PD0-15 LL_PWR_EnableGPIOPullDown
-

LL_PWR_DisableGPIOPullDown

Function name

__STATIC_INLINE void LL_PWR_DisableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F
  - LL_PWR_GPIO_G

  (*) value not defined in all devices
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- PDCRA PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRB PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRC PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRD PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRE PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRF PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRG PD0-15 LL_PWR_DisableGPIOPullDown
-

LL_PWR_IsEnabledGPIOPullDown

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Check if GPIO pull-down state is enabled.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D
  - LL_PWR_GPIO_E
  - LL_PWR_GPIO_F
  - LL_PWR_GPIO_G

  (*) value not defined in all devices
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PDCRA PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRB PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRC PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRD PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRE PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRF PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRG PD0-15 LL_PWR_IsEnabledGPIOPullDown
-

LL_PWR_IsActiveFlag_InternWU

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_InternWU (void)

Function description

Get Internal Wake-up line Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUFI LL_PWR_IsActiveFlag_InternWU

LL_PWR_IsActiveFlag_SB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)

Function description

Get Stand-By Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_WU5

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU5 (void)

Function description

Get Wake-up Flag 5.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF5 LL_PWR_IsActiveFlag_WU5

LL_PWR_IsActiveFlag_WU4

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU4 (void)

Function description

Get Wake-up Flag 4.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF4 LL_PWR_IsActiveFlag_WU4

LL_PWR_IsActiveFlag_WU3

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU3 (void)

Function description

Get Wake-up Flag 3.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF3 LL_PWR_IsActiveFlag_WU3

LL_PWR_IsActiveFlag_WU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU2 (void)

Function description

Get Wake-up Flag 2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF2 LL_PWR_IsActiveFlag_WU2

LL_PWR_IsActiveFlag_WU1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU1 (void)

Function description

Get Wake-up Flag 1.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF1 LL_PWR_IsActiveFlag_WU1

LL_PWR_ClearFlag_SB

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)

Function description

Clear Stand-By Flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CSBF LL_PWR_ClearFlag_SB

LL_PWR_ClearFlag_WU

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)

Function description

Clear Wake-up Flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF LL_PWR_ClearFlag_WU

LL_PWR_ClearFlag_WU5

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU5 (void)

Function description

Clear Wake-up Flag 5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF5 LL_PWR_ClearFlag_WU5

LL_PWR_ClearFlag_WU4

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU4 (void)

Function description

Clear Wake-up Flag 4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF4 LL_PWR_ClearFlag_WU4

LL_PWR_ClearFlag_WU3

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU3 (void)

Function description

Clear Wake-up Flag 3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF3 LL_PWR_ClearFlag_WU3

LL_PWR_ClearFlag_WU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU2 (void)

Function description

Clear Wake-up Flag 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF2 LL_PWR_ClearFlag_WU2

LL_PWR_ClearFlag_WU1

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU1 (void)

Function description

Clear Wake-up Flag 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF1 LL_PWR_ClearFlag_WU1

LL_PWR_IsActiveFlag_PVMO4

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO4 (void)

Function description

Indicate whether VDDA voltage is below or above PVM4 threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 PVMO4 LL_PWR_IsActiveFlag_PVMO4

LL_PWR_IsActiveFlag_PVMO3

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO3 (void)

Function description

Indicate whether VDDA voltage is below or above PVM3 threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 PVMO3 LL_PWR_IsActiveFlag_PVMO3

LL_PWR_IsActiveFlag_PVMO2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO2 (void)

Function description

Indicate whether VDDIO2 voltage is below or above PVM2 threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 PVMO2 LL_PWR_IsActiveFlag_PVMO2

LL_PWR_IsActiveFlag_PVMO1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVMO1 (void)

Function description

Indicate whether VDDUSB voltage is below or above PVM1 threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 PVMO1 LL_PWR_IsActiveFlag_PVMO1

LL_PWR_IsActiveFlag_PVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)

Function description

Indicate whether VDD voltage is below or above the selected PVD threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_VOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOS (void)

Function description

Indicate whether the regulator is ready in the selected voltage range or if its output voltage is still changing to the required voltage level.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 VOSF LL_PWR_IsActiveFlag_VOS

LL_PWR_IsActiveFlag_REGLPF

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPF (void)

Function description

Indicate whether the regulator is ready in main mode or is in low-power mode.

Return values

- **State:** of bit (1 or 0).

Notes

- : Take care, return value "0" means the regulator is ready. Return value "1" means the output voltage range is still changing.

Reference Manual to LL API cross reference:

- SR2 REGLPF LL_PWR_IsActiveFlag_REGLPF

LL_PWR_IsActiveFlag_REGLPS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPS (void)

Function description

Indicate whether or not the low-power regulator is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 REGLPS LL_PWR_IsActiveFlag_REGLPS

LL_PWR_DeInit

Function name

ErrorStatus LL_PWR_DeInit (void)

Function description

De-initialize the PWR registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: PWR registers are de-initialized
  - ERROR: not applicable

84.2 PWR Firmware driver defines

The following section lists the various define and macros of the module.

84.2.1 PWR

PWR

BATT CHARG RESISTOR

LL_PWR_BATT_CHARG_RESISTOR_5K

LL_PWR_BATT_CHARGRESISTOR_1_5K

Clear Flags Defines

LL_PWR_SCR_CSBF

LL_PWR_SCR_CWUF

LL_PWR_SCR_CWUF5

LL_PWR_SCR_CWUF4

LL_PWR_SCR_CWUF3

LL_PWR_SCR_CWUF2

LL_PWR_SCR_CWUF1

Get Flags Defines

LL_PWR_SR1_WUFI

LL_PWR_SR1_SBF

LL_PWR_SR1_WUF5

LL_PWR_SR1_WUF4

LL_PWR_SR1_WUF3

LL_PWR_SR1_WUF2

LL_PWR_SR1_WUF1

LL_PWR_SR2_PVMO4

LL_PWR_SR2_PVMO3

LL_PWR_SR2_PVMO2

LL_PWR_SR2_PVMO1

LL_PWR_SR2_PVDO

LL_PWR_SR2_VOSF

LL_PWR_SR2_REGLPF

LL_PWR_SR2_REGLPS

GPIO

LL_PWR_GPIO_A

LL_PWR_GPIO_B

LL_PWR_GPIO_C

LL_PWR_GPIO_D

LL_PWR_GPIO_E

LL_PWR_GPIO_F

LL_PWR_GPIO_G

GPIO BIT

LL_PWR_GPIO_BIT_0

LL_PWR_GPIO_BIT_1

LL_PWR_GPIO_BIT_2

LL_PWR_GPIO_BIT_3

LL_PWR_GPIO_BIT_4

LL_PWR_GPIO_BIT_5

LL_PWR_GPIO_BIT_6

LL_PWR_GPIO_BIT_7

LL_PWR_GPIO_BIT_8

LL_PWR_GPIO_BIT_9

LL_PWR_GPIO_BIT_10

LL_PWR_GPIO_BIT_11

LL_PWR_GPIO_BIT_12

LL_PWR_GPIO_BIT_13

LL_PWR_GPIO_BIT_14

LL_PWR_GPIO_BIT_15

MODE PWR

LL_PWR_MODE_STOP0

LL_PWR_MODE_STOP1

LL_PWR_MODE_STANDBY

LL_PWR_MODE_SHUTDOWN

PVDLEVEL

LL_PWR_PVDLEVEL_0

LL_PWR_PVDLEVEL_1

LL_PWR_PVDLEVEL_2

LL_PWR_PVDLEVEL_3

LL_PWR_PVDLEVEL_4

LL_PWR_PVDLEVEL_5

LL_PWR_PVDLEVEL_6

LL_PWR_PVDLEVEL_7

Peripheral voltage monitoring

LL_PWR_PVM_VDDA_COMP

LL_PWR_PVM_VDDA_FASTDAC

LL_PWR_PVM_VDDA_ADC

LL_PWR_PVM_VDDA_OPAMP_DAC

REGU VOLTAGE

LL_PWR_REGU_VOLTAGE_SCALE1

LL_PWR_REGU_VOLTAGE_SCALE2

WAKEUP

LL_PWR_WAKEUP_PIN1

LL_PWR_WAKEUP_PIN2

LL_PWR_WAKEUP_PIN3

LL_PWR_WAKEUP_PIN4

LL_PWR_WAKEUP_PIN5

Legacy functions name

LL_PWR_IsActiveFlag_VOSF

LL_PWR_EnableUSBDeadBattery

LL_PWR_DisableUSBDeadBattery

LL_PWR_IsEnabledUSBDeadBattery

LL_PWR_EnableDeadBatteryPD

LL_PWR_DisableDeadBatteryPD

LL_PWR_EnableUSBStandByModePD

LL_PWR_EnableStandByModePD

LL_PWR_DisableUSBStandByModePD

LL_PWR_DisableStandByModePD

LL_PWR_IsEnabledUSBStandByModePD

Common Write and read registers Macros

LL_PWR_WriteReg

Description:

- Write a value in PWR register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PWR_ReadReg

Description:

- Read a value in PWR register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

85 LL RCC Generic Driver

85.1 RCC Firmware driver registers structures

85.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32g4xx_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
  HCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
  PCLK1 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
  PCLK2 clock frequency

85.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

85.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)

Function description

Enable the Clock Security System.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSSON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)

Function description

Enable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)

Function description

Disable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name

__STATIC_INLINE void LL_RCC_HSE_Enable (void)

Function description

Enable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name

__STATIC_INLINE void LL_RCC_HSE_Disable (void)

Function description

Disable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)

Function description

Check if HSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSI_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void)

Function description

Enable HSI even in stop mode.

Return values

- **None:**

Notes

- HSI oscillator is forced ON even in Stop mode

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_EnableInStopMode

LL_RCC_HSI_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void)

Function description

Disable HSI in stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_DisableInStopMode

LL_RCC_HSI_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI_Enable (void)

Function description

Enable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI_Disable (void)

Function description

Disable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)

Function description

Check if HSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)

Function description

Get HSI Calibration value.

Return values

- **Between:** Min_Data = 0x00 and Max_Data = 0xFF

Notes

- When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- ICSCR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)

Function description

Set HSI Calibration trimming.

Parameters

- **Value:** Between Min_Data = 0 and Max_Data = 127

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the HSICAL
- Default value is 16, which, when added to the HSICAL value, should trim the HSI to 16 MHz +/- 1 %

Reference Manual to LL API cross reference:

- ICSCR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)

Function description

Get HSI Calibration trimming.

Return values

- **Between:** Min_Data = 0 and Max_Data = 127

Reference Manual to LL API cross reference:

- ICSCR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_HSI48_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Enable (void)

Function description

Enable HSI48.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRCR HSI48ON LL_RCC_HSI48_Enable

LL_RCC_HSI48_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Disable (void)

Function description

Disable HSI48.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRCR HSI48ON LL_RCC_HSI48_Disable

LL_RCC_HSI48_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)

Function description

Check if HSI48 oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CRRCR HSI48RDY LL_RCC_HSI48_IsReady

LL_RCC_HSI48_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void)

Function description

Get HSI48 Calibration value.

Return values

- **Between:** Min_Data = 0x00 and Max_Data = 0x1FF

Reference Manual to LL API cross reference:

- CRRCR HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_LSE_Enable

Function name

__STATIC_INLINE void LL_RCC_LSE_Enable (void)

Function description

Enable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name

__STATIC_INLINE void LL_RCC_LSE_Disable (void)

Function description

Disable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)

Function description

Enable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)

Function description

Disable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetDriveCapability

Function name

__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)

Function description

Set LSE oscillator drive capability.

Parameters

- **LSEDrive:** This parameter can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Return values

- **None:**

Notes

- The oscillator is in Xtal mode when it is not in bypass mode.

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)

Function description

Get LSE oscillator drive capability.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)

Function description

Enable Clock security system on LSE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_DisableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void)

Function description

Disable Clock security system on LSE.

Return values

- **None:**

Notes

- Clock security system can be disabled only after a LSE failure detection. In that case it MUST be disabled by software.

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_DisableCSS

LL_RCC_LSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)

Function description

Check if LSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSE_IsCSSDetected

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsCSSDetected (void)

Function description

Check if CSS on LSE failure Detection.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSECSSD LL_RCC_LSE_IsCSSDetected

LL_RCC_LSI_Enable

Function name

__STATIC_INLINE void LL_RCC_LSI_Enable (void)

Function description

Enable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR LSION LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name

__STATIC_INLINE void LL_RCC_LSI_Disable (void)

Function description

Disable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)

Function description

Check if LSI is Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_LSCO_Enable

Function name

__STATIC_INLINE void LL_RCC_LSCO_Enable (void)

Function description

Enable Low speed clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOEN LL_RCC_LSCO_Enable

LL_RCC_LSCO_Disable

Function name

__STATIC_INLINE void LL_RCC_LSCO_Disable (void)

Function description

Disable Low speed clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOEN LL_RCC_LSCO_Disable

LL_RCC_LSCO_SetSource

Function name

__STATIC_INLINE void LL_RCC_LSCO_SetSource (uint32_t Source)

Function description

Configure Low speed clock selection.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_LSCO_CLKSOURCE_LSI
  - LL_RCC_LSCO_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOSEL LL_RCC_LSCO_SetSource

LL_RCC_LSCO_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_LSCO_GetSource (void)

Function description

Get Low speed clock selection.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSCO_CLKSOURCE_LSI
  - LL_RCC_LSCO_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- BDCR LSCOSEL LL_RCC_LSCO_GetSource

LL_RCC_SetSysClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)

Function description

Configure the system clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_HSI
  - LL_RCC_SYS_CLKSOURCE_HSE
  - LL_RCC_SYS_CLKSOURCE_PLL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)

Function description

Get the system clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_PLL

Reference Manual to LL API cross reference:

- CFGR SWS LL_RCC_GetSysClkSource

LL_RCC_SetAHBPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)

Function description

Set AHB prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetAPB1Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)

Function description

Set APB1 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR PPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)

Function description

Set APB2 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_GetAHBPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)

Function description

Get AHB prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Reference Manual to LL API cross reference:

- CFGR HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetAPB1Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)

Function description

Get APB1 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Reference Manual to LL API cross reference:

- CFGR PPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_GetAPB2Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)

Function description

Get APB2 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Reference Manual to LL API cross reference:

- CFGR PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_ConfigMCO

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)

Function description

Configure MCOx.

Parameters

- **MCOxSource:** This parameter can be one of the following values:
  - LL_RCC_MCO1SOURCE_NOCLOCK
  - LL_RCC_MCO1SOURCE_SYSCLK
  - LL_RCC_MCO1SOURCE_HSI
  - LL_RCC_MCO1SOURCE_HSE
  - LL_RCC_MCO1SOURCE_HSI48
  - LL_RCC_MCO1SOURCE_PLLCLK
  - LL_RCC_MCO1SOURCE_LSI
  - LL_RCC_MCO1SOURCE_LSE

  (*) value not defined in all devices.
- **MCOxPrescaler:** This parameter can be one of the following values:
  - LL_RCC_MCO1_DIV_1
  - LL_RCC_MCO1_DIV_2
  - LL_RCC_MCO1_DIV_4
  - LL_RCC_MCO1_DIV_8
  - LL_RCC_MCO1_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR MCOSEL LL_RCC_ConfigMCO
- CFGR MCOPRE LL_RCC_ConfigMCO

LL_RCC_SetUSARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t USARTxSource)

Function description

Configure USARTx clock source.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_SYSCLK
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_SYSCLK
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_SYSCLK
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR USARTxSEL LL_RCC_SetUSARTClockSource

LL_RCC_SetUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUARTClockSource (uint32_t UARTxSource)

Function description

Configure UARTx clock source.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART4_CLKSOURCE_SYSCLK (*)
  - LL_RCC_UART4_CLKSOURCE_HSI (*)
  - LL_RCC_UART4_CLKSOURCE_LSE (*)
  - LL_RCC_UART5_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART5_CLKSOURCE_SYSCLK (*)
  - LL_RCC_UART5_CLKSOURCE_HSI (*)
  - LL_RCC_UART5_CLKSOURCE_LSE (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR UARTxSEL LL_RCC_SetUARTClockSource

LL_RCC_SetLPUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t LPUARTxSource)

Function description

Configure LPUART1x clock source.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK1
  - LL_RCC_LPUART1_CLKSOURCE_SYSCLK
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetI2CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t I2CxSource)

Function description

Configure I2Cx clock source.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_SYSCLK
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_SYSCLK
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK1
  - LL_RCC_I2C3_CLKSOURCE_SYSCLK
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1 (*)
  - LL_RCC_I2C4_CLKSOURCE_SYSCLK (*)
  - LL_RCC_I2C4_CLKSOURCE_HSI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR I2CxSEL LL_RCC_SetI2CClockSource

LL_RCC_SetLPTIMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t LPTIMxSource)

Function description

Configure LPTIMx clock source.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_HSI
  - LL_RCC_LPTIM1_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR LPTIM1SEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetSAIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSAIClockSource (uint32_t SAIxSource)

Function description

Configure SAIx clock source.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_SYSCLK
  - LL_RCC_SAI1_CLKSOURCE_PLL
  - LL_RCC_SAI1_CLKSOURCE_PIN
  - LL_RCC_SAI1_CLKSOURCE_HSI

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR SAI1SEL LL_RCC_SetSAIClockSource

LL_RCC_SetI2SClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2SClockSource (uint32_t I2SxSource)

Function description

Configure I2S clock source.

Parameters

- **I2SxSource:** This parameter can be one of the following values:
  - LL_RCC_I2S_CLKSOURCE_SYSCLK
  - LL_RCC_I2S_CLKSOURCE_PLL
  - LL_RCC_I2S_CLKSOURCE_PIN
  - LL_RCC_I2S_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR I2S23SEL LL_RCC_SetI2SClockSource

LL_RCC_SetFDCANClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFDCANClockSource (uint32_t FDCANxSource)

Function description

Configure FDCAN clock source.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL
  - LL_RCC_FDCAN_CLKSOURCE_PCLK1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR FDCANSEL LL_RCC_SetFDCANClockSource

LL_RCC_SetRNGClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t RNGxSource)

Function description

Configure RNG clock source.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_PLL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR CLK48SEL LL_RCC_SetRNGClockSource

LL_RCC_SetUSBClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)

Function description

Configure USB clock source.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_HSI48
  - LL_RCC_USB_CLKSOURCE_PLL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR CLK48SEL LL_RCC_SetUSBClockSource

LL_RCC_SetADCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADCClockSource (uint32_t ADCxSource)

Function description

Configure ADC clock source.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADC12_CLKSOURCE_NONE
  - LL_RCC_ADC12_CLKSOURCE_PLL
  - LL_RCC_ADC12_CLKSOURCE_SYSCLK
  - LL_RCC_ADC345_CLKSOURCE_NONE (*)
  - LL_RCC_ADC345_CLKSOURCE_PLL (*)
  - LL_RCC_ADC345_CLKSOURCE_SYSCLK (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR ADC12SEL LL_RCC_SetADCClockSource
- CCIPR ADC345SEL LL_RCC_SetADCClockSource

LL_RCC_SetQUADSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetQUADSPIClockSource (uint32_t Source)

Function description

Configure QUADSPI clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_QUADSPI_CLKSOURCE_SYSCLK
  - LL_RCC_QUADSPI_CLKSOURCE_HSI
  - LL_RCC_QUADSPI_CLKSOURCE_PLL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 QSPISEL LL_RCC_SetQUADSPIClockSource

LL_RCC_GetUSARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTx)

Function description

Get USARTx clock source.

Parameters

- **USARTx:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_SYSCLK
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_SYSCLK
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_SYSCLK
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR USARTxSEL LL_RCC_GetUSARTClockSource

LL_RCC_GetUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUARTClockSource (uint32_t UARTx)

Function description

Get UARTx clock source.

Parameters

- **UARTx:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE (*)
  - LL_RCC_UART5_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART4_CLKSOURCE_SYSCLK (*)
  - LL_RCC_UART4_CLKSOURCE_HSI (*)
  - LL_RCC_UART4_CLKSOURCE_LSE (*)
  - LL_RCC_UART5_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART5_CLKSOURCE_SYSCLK (*)
  - LL_RCC_UART5_CLKSOURCE_HSI (*)
  - LL_RCC_UART5_CLKSOURCE_LSE (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- CCIPR UARTxSEL LL_RCC_GetUARTClockSource

LL_RCC_GetLPUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t LPUARTx)

Function description

Get LPUARTx clock source.

Parameters

- **LPUARTx:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK1
  - LL_RCC_LPUART1_CLKSOURCE_SYSCLK
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetI2CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)

Function description

Get I2Cx clock source.

Parameters

- **I2Cx:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE (*)

  (*) value not defined in all devices.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_SYSCLK
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_SYSCLK
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK1
  - LL_RCC_I2C3_CLKSOURCE_SYSCLK
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1 (*)
  - LL_RCC_I2C4_CLKSOURCE_SYSCLK (*)
  - LL_RCC_I2C4_CLKSOURCE_HSI (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- CCIPR I2CxSEL LL_RCC_GetI2CClockSource

LL_RCC_GetLPTIMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t LPTIMx)

Function description

Get LPTIMx clock source.

Parameters

- **LPTIMx:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_HSI
  - LL_RCC_LPTIM1_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR LPTIMxSEL LL_RCC_GetLPTIMClockSource

LL_RCC_GetSAIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSAIClockSource (uint32_t SAIx)

Function description

Get SAIx clock source.

Parameters

- **SAIx:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE

  (*) value not defined in all devices.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_SYSCLK
  - LL_RCC_SAI1_CLKSOURCE_PLL
  - LL_RCC_SAI1_CLKSOURCE_PIN
  - LL_RCC_SAI1_CLKSOURCE_HSI

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- CCIPR SAI1SEL LL_RCC_GetSAIClockSource

LL_RCC_GetI2SClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2SClockSource (uint32_t I2Sx)

Function description

Get I2Sx clock source.

Parameters

- **I2Sx:** This parameter can be one of the following values:
  - LL_RCC_I2S_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2S_CLKSOURCE_SYSCLK
  - LL_RCC_I2S_CLKSOURCE_PLL
  - LL_RCC_I2S_CLKSOURCE_PIN
  - LL_RCC_I2S_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR I2S23SEL LL_RCC_GetI2SClockSource

LL_RCC_GetFDCANClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFDCANClockSource (uint32_t FDCANx)

Function description

Get FDCANx clock source.

Parameters

- **FDCANx:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL
  - LL_RCC_FDCAN_CLKSOURCE_PCLK1
- **None:**

Reference Manual to LL API cross reference:

- CCIPR FDCANSEL LL_RCC_GetFDCANClockSource

LL_RCC_GetRNGClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t RNGx)

Function description

Get RNGx clock source.

Parameters

- **RNGx:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_PLL

Reference Manual to LL API cross reference:

- CCIPR CLK48SEL LL_RCC_GetRNGClockSource

LL_RCC_GetUSBClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t USBx)

Function description

Get USBx clock source.

Parameters

- **USBx:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_HSI48
  - LL_RCC_USB_CLKSOURCE_PLL

Reference Manual to LL API cross reference:

- CCIPR CLK48SEL LL_RCC_GetUSBClockSource

LL_RCC_GetADCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCClockSource (uint32_t ADCx)

Function description

Get ADCx clock source.

Parameters

- **ADCx:** This parameter can be one of the following values:
  - LL_RCC_ADC12_CLKSOURCE
  - LL_RCC_ADC345_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ADC12_CLKSOURCE_NONE
  - LL_RCC_ADC12_CLKSOURCE_PLL
  - LL_RCC_ADC12_CLKSOURCE_SYSCLK
  - LL_RCC_ADC345_CLKSOURCE_NONE (*)
  - LL_RCC_ADC345_CLKSOURCE_PLL (*)
  - LL_RCC_ADC345_CLKSOURCE_SYSCLK (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- CCIPR ADCSEL LL_RCC_GetADCClockSource

LL_RCC_GetQUADSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetQUADSPIClockSource (uint32_t QUADSPIx)

Function description

Get QUADSPI clock source.

Parameters

- **QUADSPIx:** This parameter can be one of the following values:
  - LL_RCC_QUADSPI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_QUADSPI_CLKSOURCE_SYSCLK
  - LL_RCC_QUADSPI_CLKSOURCE_HSI
  - LL_RCC_QUADSPI_CLKSOURCE_PLL

Reference Manual to LL API cross reference:

- CCIPR2 QSPISEL LL_RCC_GetQUADSPIClockSource

LL_RCC_SetRTCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)

Function description

Set RTC Clock Source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV32

Return values

- **None:**

Notes

- Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The BDRST bit can be used to reset them.

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)

Function description

Get RTC Clock Source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV32

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name

__STATIC_INLINE void LL_RCC_EnableRTC (void)

Function description

Enable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name

__STATIC_INLINE void LL_RCC_DisableRTC (void)

Function description

Disable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)

Function description

Check if RTC has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)

Function description

Force the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)

Function description

Release the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_PLL_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL_Enable (void)

Function description

Enable PLL.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLLON LL_RCC_PLL_Enable

LL_RCC_PLL_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL_Disable (void)

Function description

Disable PLL.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL clock is used as the system clock

Reference Manual to LL API cross reference:

- CR PLLON LL_RCC_PLL_Disable

LL_RCC_PLL_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_IsReady (void)

Function description

Check if PLL Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLLRDY LL_RCC_PLL_IsReady

LL_RCC_PLL_ConfigDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SYS (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)

Function description

Configure PLL used for SYSCLK Domain.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_NONE
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_HSE
- **PLLM:** This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16
- **PLLN:** Between Min_Data = 8 and Max_Data = 127
- **PLLR:** This parameter can be one of the following values:
  - LL_RCC_PLLR_DIV_2
  - LL_RCC_PLLR_DIV_4
  - LL_RCC_PLLR_DIV_6
  - LL_RCC_PLLR_DIV_8

Return values

- **None:**

Notes

- PLL Source and PLLM Divider can be written only when PLL is disabled.
- PLLN/PLLR can be written only when PLL is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL_RCC_PLL_ConfigDomain_SYS
- PLLCFGR PLLM LL_RCC_PLL_ConfigDomain_SYS
- PLLCFGR PLLN LL_RCC_PLL_ConfigDomain_SYS
- PLLCFGR PLLR LL_RCC_PLL_ConfigDomain_SYS

LL_RCC_PLL_ConfigDomain_ADC

Function name

__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_ADC (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)

Function description

Configure PLL used for ADC domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_NONE
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_HSE
- **PLLM:** This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16
- **PLLN:** Between Min_Data = 8 and Max_Data = 127

- **PLLP:** This parameter can be one of the following values:
  - LL_RCC_PLLP_DIV_2
  - LL_RCC_PLLP_DIV_3
  - LL_RCC_PLLP_DIV_4
  - LL_RCC_PLLP_DIV_5
  - LL_RCC_PLLP_DIV_6
  - LL_RCC_PLLP_DIV_7
  - LL_RCC_PLLP_DIV_8
  - LL_RCC_PLLP_DIV_9
  - LL_RCC_PLLP_DIV_10
  - LL_RCC_PLLP_DIV_11
  - LL_RCC_PLLP_DIV_12
  - LL_RCC_PLLP_DIV_13
  - LL_RCC_PLLP_DIV_14
  - LL_RCC_PLLP_DIV_15
  - LL_RCC_PLLP_DIV_16
  - LL_RCC_PLLP_DIV_17
  - LL_RCC_PLLP_DIV_18
  - LL_RCC_PLLP_DIV_19
  - LL_RCC_PLLP_DIV_20
  - LL_RCC_PLLP_DIV_21
  - LL_RCC_PLLP_DIV_22
  - LL_RCC_PLLP_DIV_23
  - LL_RCC_PLLP_DIV_24
  - LL_RCC_PLLP_DIV_25
  - LL_RCC_PLLP_DIV_26
  - LL_RCC_PLLP_DIV_27
  - LL_RCC_PLLP_DIV_28
  - LL_RCC_PLLP_DIV_29
  - LL_RCC_PLLP_DIV_30
  - LL_RCC_PLLP_DIV_31

Return values

- **None:**

Notes

- PLL Source and PLLM Divider can be written only when PLL is disabled.
- PLLN/PLLP can be written only when PLL is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL_RCC_PLL_ConfigDomain_ADC
- PLLCFGR PLLM LL_RCC_PLL_ConfigDomain_ADC
- PLLCFGR PLLN LL_RCC_PLL_ConfigDomain_ADC
- PLLCFGR PLLPDIV LL_RCC_PLL_ConfigDomain_ADC

LL_RCC_PLL_ConfigDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)

Function description

Configure PLL used for 48Mhz domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_NONE
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_HSE
- **PLLM:** This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16
- **PLLN:** Between Min_Data = 8 and Max_Data = 127
- **PLLQ:** This parameter can be one of the following values:
  - LL_RCC_PLLQ_DIV_2
  - LL_RCC_PLLQ_DIV_4
  - LL_RCC_PLLQ_DIV_6
  - LL_RCC_PLLQ_DIV_8

Return values

- **None:**

Notes

- PLL Source and PLLM Divider can be written only when PLL, is disabled.
- PLLN/PLLQ can be written only when PLL is disabled.
- This can be selected for USB, RNG

Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL_RCC_PLL_ConfigDomain_48M
- PLLCFGR PLLM LL_RCC_PLL_ConfigDomain_48M
- PLLCFGR PLLN LL_RCC_PLL_ConfigDomain_48M
- PLLCFGR PLLQ LL_RCC_PLL_ConfigDomain_48M

LL_RCC_PLL_SetMainSource

Function name

__STATIC_INLINE void LL_RCC_PLL_SetMainSource (uint32_t PLLSource)

Function description

Configure PLL clock source.

Parameters

- **PLLSource:** This parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_NONE
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL_RCC_PLL_SetMainSource

LL_RCC_PLL_GetMainSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetMainSource (void)

Function description

Get the oscillator used as PLL clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLSOURCE_NONE
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_HSE

Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL_RCC_PLL_GetMainSource

LL_RCC_PLL_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetN (void)

Function description

Get Main PLL multiplication factor for VCO.

Return values

- **Between:** Min_Data = 8 and Max_Data = 127

Reference Manual to LL API cross reference:

- PLLCFGR PLLN LL_RCC_PLL_GetN

LL_RCC_PLL_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetP (void)

Function description

Get Main PLL division factor for PLLP.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLP_DIV_2
  - LL_RCC_PLLP_DIV_3
  - LL_RCC_PLLP_DIV_4
  - LL_RCC_PLLP_DIV_5
  - LL_RCC_PLLP_DIV_6
  - LL_RCC_PLLP_DIV_7
  - LL_RCC_PLLP_DIV_8
  - LL_RCC_PLLP_DIV_9
  - LL_RCC_PLLP_DIV_10
  - LL_RCC_PLLP_DIV_11
  - LL_RCC_PLLP_DIV_12
  - LL_RCC_PLLP_DIV_13
  - LL_RCC_PLLP_DIV_14
  - LL_RCC_PLLP_DIV_15
  - LL_RCC_PLLP_DIV_16
  - LL_RCC_PLLP_DIV_17
  - LL_RCC_PLLP_DIV_18
  - LL_RCC_PLLP_DIV_19
  - LL_RCC_PLLP_DIV_20
  - LL_RCC_PLLP_DIV_21
  - LL_RCC_PLLP_DIV_22
  - LL_RCC_PLLP_DIV_23
  - LL_RCC_PLLP_DIV_24
  - LL_RCC_PLLP_DIV_25
  - LL_RCC_PLLP_DIV_26
  - LL_RCC_PLLP_DIV_27
  - LL_RCC_PLLP_DIV_28
  - LL_RCC_PLLP_DIV_29
  - LL_RCC_PLLP_DIV_30
  - LL_RCC_PLLP_DIV_31

Notes

- Used for PLLADCCLK (ADC clock)

Reference Manual to LL API cross reference:

- PLLCFGR PLLPDIV LL_RCC_PLL_GetP
-
- PLLCFGR PLLP LL_RCC_PLL_GetP

LL_RCC_PLL_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetQ (void)

Function description

Get Main PLL division factor for PLLQ.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLQ_DIV_2
  - LL_RCC_PLLQ_DIV_4
  - LL_RCC_PLLQ_DIV_6
  - LL_RCC_PLLQ_DIV_8

Notes

- Used for PLL48M1CLK selected for USB, RNG (48 MHz clock)

Reference Manual to LL API cross reference:

- PLLCFGR PLLQ LL_RCC_PLL_GetQ

LL_RCC_PLL_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetR (void)

Function description

Get Main PLL division factor for PLLR.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLR_DIV_2
  - LL_RCC_PLLR_DIV_4
  - LL_RCC_PLLR_DIV_6
  - LL_RCC_PLLR_DIV_8

Notes

- Used for PLLCLK (system clock)

Reference Manual to LL API cross reference:

- PLLCFGR PLLR LL_RCC_PLL_GetR

LL_RCC_PLL_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetDivider (void)

Function description

Get Division factor for the main PLL and other PLL.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16

Reference Manual to LL API cross reference:

- PLLCFGR PLLM LL_RCC_PLL_GetDivider

LL_RCC_PLL_EnableDomain_ADC

Function name

__STATIC_INLINE void LL_RCC_PLL_EnableDomain_ADC (void)

Function description

Enable PLL output mapped on ADC domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLLPEN LL_RCC_PLL_EnableDomain_ADC

LL_RCC_PLL_DisableDomain_ADC

Function name

__STATIC_INLINE void LL_RCC_PLL_DisableDomain_ADC (void)

Function description

Disable PLL output mapped on ADC domain clock.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL clock is used as the system clock
- In order to save power, when the PLLCLK of the PLL is not used, should be 0

Reference Manual to LL API cross reference:

- PLLCFGR PLLPEN LL_RCC_PLL_DisableDomain_ADC

LL_RCC_PLL_EnableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL_EnableDomain_48M (void)

Function description

Enable PLL output mapped on 48MHz domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLLQEN LL_RCC_PLL_EnableDomain_48M

LL_RCC_PLL_DisableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL_DisableDomain_48M (void)

Function description

Disable PLL output mapped on 48MHz domain clock.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL clock is used as the system clock
- In order to save power, when the PLLCLK of the PLL is not used, should be 0

Reference Manual to LL API cross reference:

- PLLCFGR PLLQEN LL_RCC_PLL_DisableDomain_48M

LL_RCC_PLL_EnableDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL_EnableDomain_SYS (void)

Function description

Enable PLL output mapped on SYSCLK domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLLREN LL_RCC_PLL_EnableDomain_SYS

LL_RCC_PLL_DisableDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL_DisableDomain_SYS (void)

Function description

Disable PLL output mapped on SYSCLK domain.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL clock is used as the system clock
- In order to save power, when the PLLCLK of the PLL is not used, Main PLL should be 0

Reference Manual to LL API cross reference:

- PLLCFGR PLLREN LL_RCC_PLL_DisableDomain_SYS

LL_RCC_ClearFlag_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)

Function description

Clear LSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)

Function description

Clear LSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)

Function description

Clear HSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)

Function description

Clear HSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_PLLRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLLRDY (void)

Function description

Clear PLL ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLLRDYC LL_RCC_ClearFlag_PLLRDY

LL_RCC_ClearFlag_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)

Function description

Clear HSI48 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_HSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)

Function description

Clear Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR CSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_ClearFlag_LSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void)

Function description

Clear LSE Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSECSSC LL_RCC_ClearFlag_LSECSS

LL_RCC_IsActiveFlag_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)

Function description

Check if LSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)

Function description

Check if LSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)

Function description

Check if HSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)

Function description

Check if HSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_PLLRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLRDY (void)

Function description

Check if PLL ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLLRDYF LL_RCC_IsActiveFlag_PLLRDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)

Function description

Check if HSI48 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_HSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)

Function description

Check if Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR CSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS (void)

Function description

Check if LSE Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSECSSF LL_RCC_IsActiveFlag_LSECSS

LL_RCC_IsActiveFlag_IWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)

Function description

Check if RCC flag Independent Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC flag Low Power reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_OBLRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_OBLRST (void)

Function description

Check if RCC flag Option byte reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OBLRSTF LL_RCC_IsActiveFlag_OBLRST

LL_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_WWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)

Function description

Check if RCC flag Window Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR BORRSTF LL_RCC_IsActiveFlag_BORRST

LL_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RMVF LL_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)

Function description

Enable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)

Function description

Enable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)

Function description

Enable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)

Function description

Enable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_PLLRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLLRDY (void)

Function description

Enable PLL ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLLRDYIE LL_RCC_EnableIT_PLLRDY

LL_RCC_EnableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)

Function description

Enable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

LL_RCC_EnableIT_LSECSS

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSECSS (void)

Function description

Enable LSE clock security system interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_EnableIT_LSECSS

LL_RCC_DisableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)

Function description

Disable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)

Function description

Disable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)

Function description

Disable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)

Function description

Disable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_PLLRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLLRDY (void)

Function description

Disable PLL ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLLRDYIE LL_RCC_DisableIT_PLLRDY

LL_RCC_DisableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)

Function description

Disable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

LL_RCC_DisableIT_LSECSS

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSECSS (void)

Function description

Disable LSE clock security system interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_DisableIT_LSECSS

LL_RCC_IsEnabledIT_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)

Function description

Checks if LSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)

Function description

Checks if LSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)

Function description

Checks if HSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

LL_RCC_IsEnabledIT_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)

Function description

Checks if HSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_PLLRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLRDY (void)

Function description

Checks if PLL ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLLRDYIE LL_RCC_IsEnabledIT_PLLRDY

LL_RCC_IsEnabledIT_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY (void)

Function description

Checks if HSI48 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

LL_RCC_IsEnabledIT_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSECSS (void)

Function description

Checks if LSECSS interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_IsEnabledIT_LSECSS

LL_RCC_DeInit

Function name

ErrorStatus LL_RCC_DeInit (void)

Function description

Reset the RCC clock configuration to the default reset state.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RCC registers are de-initialized
  - ERROR: not applicable

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)

Function description

Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSARTClockFreq

Function name

uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)

Function description

Return USARTx clock frequency.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE

Return values

- **USART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetUARTClockFreq

Function name

uint32_t LL_RCC_GetUARTClockFreq (uint32_t UARTxSource)

Function description

Return UARTx clock frequency.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE (*)
  - LL_RCC_UART5_CLKSOURCE (*)

  (*) value not defined in all devices.

Return values

- **UART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetI2CClockFreq

Function name

uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)

Function description

Return I2Cx clock frequency.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE (*)

  (*) value not defined in all devices.

Return values

- **I2C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that HSI oscillator is not ready

LL_RCC_GetLPUARTClockFreq

Function name

uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)

Function description

Return LPUARTx clock frequency.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **LPUART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetLPTIMClockFreq

Function name

uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)

Function description

Return LPTIMx clock frequency.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE

Return values

- **LPTIM:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI, LSI or LSE) is not ready

LL_RCC_GetSAIClockFreq

Function name

uint32_t LL_RCC_GetSAIClockFreq (uint32_t SAIxSource)

Function description

Return SAIx clock frequency.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE

Return values

- **SAI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that PLL is not ready

LL_RCC_GetI2SClockFreq

Function name

uint32_t LL_RCC_GetI2SClockFreq (uint32_t I2SxSource)

Function description

Return I2Sx clock frequency.

Parameters

- **I2SxSource:** This parameter can be one of the following values:
  - LL_RCC_I2S_CLKSOURCE

Return values

- **I2S:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetFDCANClockFreq

Function name

uint32_t LL_RCC_GetFDCANClockFreq (uint32_t FDCANxSource)

Function description

Return FDCAN kernel clock frequency.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **FDCAN:** kernel clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetRNGClockFreq

Function name

uint32_t LL_RCC_GetRNGClockFreq (uint32_t RNGxSource)

Function description

Return RNGx clock frequency.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **RNG:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI48) or PLL is not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetUSBClockFreq

Function name

uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)

Function description

Return USBx clock frequency.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values

- **USB:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI48) or PLL is not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetADCClockFreq

Function name

uint32_t LL_RCC_GetADCClockFreq (uint32_t ADCxSource)

Function description

Return ADCx clock frequency.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADC12_CLKSOURCE
  - LL_RCC_ADC345_CLKSOURCE (*)

  (*) value not defined in all devices.

Return values

- **ADC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that PLL is not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetQUADSPIClockFreq

Function name

uint32_t LL_RCC_GetQUADSPIClockFreq (uint32_t QUADSPIxSource)

Function description

Return QUADSPI clock frequency.

Parameters

- **QUADSPIxSource:** This parameter can be one of the following values:
  - LL_RCC_QUADSPI_CLKSOURCE

Return values

- **QUADSPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that no clock is configured

85.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

85.3.1 RCC

RCC

Peripheral ADC get clock source

LL_RCC_ADC12_CLKSOURCE

ADC12 Clock source selection

LL_RCC_ADC345_CLKSOURCE

ADC345 Clock source selection

Peripheral ADC clock source selection

LL_RCC_ADC12_CLKSOURCE_NONE

No clock used as ADC12 clock source

LL_RCC_ADC12_CLKSOURCE_PLL

PLL clock used as ADC12 clock source

LL_RCC_ADC12_CLKSOURCE_SYSCLK

SYSCLK clock used as ADC12 clock source

LL_RCC_ADC345_CLKSOURCE_NONE

No clock used as ADC345 clock source

LL_RCC_ADC345_CLKSOURCE_PLL

PLL clock used as ADC345 clock source

LL_RCC_ADC345_CLKSOURCE_SYSCLK

SYSCLK clock used as ADC345 clock source

APB low-speed prescaler (APB1)

LL_RCC_APB1_DIV_1

HCLK not divided

LL_RCC_APB1_DIV_2

HCLK divided by 2

LL_RCC_APB1_DIV_4

HCLK divided by 4

LL_RCC_APB1_DIV_8

HCLK divided by 8

LL_RCC_APB1_DIV_16

HCLK divided by 16

APB high-speed prescaler (APB2)

LL_RCC_APB2_DIV_1

HCLK not divided

LL_RCC_APB2_DIV_2

HCLK divided by 2

LL_RCC_APB2_DIV_4

HCLK divided by 4

LL_RCC_APB2_DIV_8

HCLK divided by 8

LL_RCC_APB2_DIV_16

HCLK divided by 16

Clear Flags Defines

LL_RCC_CICR_LSIRDYC

LSI Ready Interrupt Clear

LL_RCC_CICR_LSERDYC

LSE Ready Interrupt Clear

LL_RCC_CICR_HSIRDYC

HSI Ready Interrupt Clear

LL_RCC_CICR_HSERDYC

HSE Ready Interrupt Clear

LL_RCC_CICR_PLLRDYC

PLL Ready Interrupt Clear

LL_RCC_CICR_HSI48RDYC

HSI48 Ready Interrupt Clear

LL_RCC_CICR_LSECSSC

LSE Clock Security System Interrupt Clear

LL_RCC_CICR_CSSC

Clock Security System Interrupt Clear

Peripheral FDCAN get clock source

LL_RCC_FDCAN_CLKSOURCE

FDCAN Clock source selection

Peripheral FDCAN clock source selection

LL_RCC_FDCAN_CLKSOURCE_HSE

HSE clock used as FDCAN clock source

LL_RCC_FDCAN_CLKSOURCE_PLL

PLL clock used as FDCAN clock source

LL_RCC_FDCAN_CLKSOURCE_PCLK1

PCLK1 clock used as FDCAN clock source

Get Flags Defines

LL_RCC_CIFR_LSIRDYF

LSI Ready Interrupt flag

LL_RCC_CIFR_LSERDYF

LSE Ready Interrupt flag

LL_RCC_CIFR_HSIRDYF

HSI Ready Interrupt flag

LL_RCC_CIFR_HSERDYF

HSE Ready Interrupt flag

LL_RCC_CIFR_PLLRDYF

PLL Ready Interrupt flag

LL_RCC_CIFR_HSI48RDYF

HSI48 Ready Interrupt flag

LL_RCC_CIFR_LSECSSF

LSE Clock Security System Interrupt flag

LL_RCC_CIFR_CSSF

Clock Security System Interrupt flag

LL_RCC_CSR_LPWRRSTF

Low-Power reset flag

LL_RCC_CSR_OBLRSTF

OBL reset flag

LL_RCC_CSR_PINRSTF

PIN reset flag

LL_RCC_CSR_SFTRSTF

Software Reset flag

LL_RCC_CSR_IWDGRSTF

Independent Watchdog reset flag

LL_RCC_CSR_WWDGRSTF

Window watchdog reset flag

LL_RCC_CSR_BORRSTF

BOR reset flag

Peripheral I2C get clock source

LL_RCC_I2C1_CLKSOURCE

I2C1 Clock source selection

LL_RCC_I2C2_CLKSOURCE

I2C2 Clock source selection

LL_RCC_I2C3_CLKSOURCE

I2C3 Clock source selection

LL_RCC_I2C4_CLKSOURCE

I2C4 Clock source selection

Peripheral I2C clock source selection

LL_RCC_I2C1_CLKSOURCE_PCLK1

PCLK1 clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_HSI

HSI clock used as I2C1 clock source

LL_RCC_I2C2_CLKSOURCE_PCLK1

PCLK1 clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_HSI

HSI clock used as I2C2 clock source

LL_RCC_I2C3_CLKSOURCE_PCLK1

PCLK1 clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_HSI

HSI clock used as I2C3 clock source

LL_RCC_I2C4_CLKSOURCE_PCLK1

PCLK1 clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_HSI

HSI clock used as I2C4 clock source

Peripheral I2S get clock source

LL_RCC_I2S_CLKSOURCE

I2S Clock source selection

Peripheral I2S clock source selection

LL_RCC_I2S_CLKSOURCE_SYSCLK

System clock used as I2S clock source

LL_RCC_I2S_CLKSOURCE_PLL

PLL clock used as I2S clock source

LL_RCC_I2S_CLKSOURCE_PIN

EXT clock used as I2S clock source

LL_RCC_I2S_CLKSOURCE_HSI

HSI clock used as I2S clock source

IT Defines

LL_RCC_CIER_LSIRDYIE

LSI Ready Interrupt Enable

LL_RCC_CIER_LSERDYIE

LSE Ready Interrupt Enable

LL_RCC_CIER_HSIRDYIE

HSI Ready Interrupt Enable

LL_RCC_CIER_HSERDYIE

HSE Ready Interrupt Enable

LL_RCC_CIER_PLLRDYIE

PLL Ready Interrupt Enable

LL_RCC_CIER_HSI48RDYIE

HSI48 Ready Interrupt Enable

LL_RCC_CIER_LSECSSIE

LSE CSS Interrupt Enable

Peripheral LPTIM get clock source

LL_RCC_LPTIM1_CLKSOURCE

LPTIM1 Clock source selection

Peripheral LPTIM clock source selection

LL_RCC_LPTIM1_CLKSOURCE_PCLK1

PCLK1 clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_LSI

LSI clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_HSI

HSI clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_LSE

LSE clock used as LPTIM1 clock source

Peripheral LPUART get clock source

LL_RCC_LPUART1_CLKSOURCE

LPUART1 Clock source selection

Peripheral LPUART clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK1

PCLK1 clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_SYSCLK

SYSCLK clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_HSI

HSI clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_LSE

LSE clock used as LPUART1 clock source

LSCO Selection

LL_RCC_LSCO_CLKSOURCE_LSI

LSI selection for low speed clock

LL_RCC_LSCO_CLKSOURCE_LSE

LSE selection for low speed clock

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW

Xtal mode lower driving capability

LL_RCC_LSEDRIVE_MEDIUMLOW

Xtal mode medium low driving capability

LL_RCC_LSEDRIVE_MEDIUMHIGH

Xtal mode medium high driving capability

LL_RCC_LSEDRIVE_HIGH

Xtal mode higher driving capability

MCO1 SOURCE selection

LL_RCC_MCO1SOURCE_NOCLOCK

MCO output disabled, no clock on MCO

LL_RCC_MCO1SOURCE_SYSCLK

SYSCLK selection as MCO1 source

LL_RCC_MCO1SOURCE_HSI

HSI16 selection as MCO1 source

LL_RCC_MCO1SOURCE_HSE

HSE selection as MCO1 source

LL_RCC_MCO1SOURCE_PLLCLK

Main PLL selection as MCO1 source

LL_RCC_MCO1SOURCE_LSI

LSI selection as MCO1 source

LL_RCC_MCO1SOURCE_LSE

LSE selection as MCO1 source

LL_RCC_MCO1SOURCE_HSI48

HSI48 selection as MCO1 source

MCO1 prescaler

LL_RCC_MCO1_DIV_1

MCO not divided

LL_RCC_MCO1_DIV_2

MCO divided by 2

LL_RCC_MCO1_DIV_4

MCO divided by 4

LL_RCC_MCO1_DIV_8

MCO divided by 8

LL_RCC_MCO1_DIV_16

MCO divided by 16

Oscillator Values adaptation

HSE_VALUE

Value of the HSE oscillator in Hz

HSI_VALUE

Value of the HSI oscillator in Hz

LSE_VALUE

Value of the LSE oscillator in Hz

LSI_VALUE

Value of the LSI oscillator in Hz

HSI48_VALUE

Value of the HSI48 oscillator in Hz

EXTERNAL_CLOCK_VALUE

Value of the I2S_CKIN, I2S and SAI1 external clock source in Hz

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO

No clock enabled for the peripheral

LL_RCC_PERIPH_FREQUENCY_NA

Frequency cannot be provided as external clock

PLL division factor

LL_RCC_PLLM_DIV_1

PLL division factor by 1

LL_RCC_PLLM_DIV_2

PLL division factor by 2

LL_RCC_PLLM_DIV_3

PLL division factor by 3

LL_RCC_PLLM_DIV_4

PLL division factor by 4

LL_RCC_PLLM_DIV_5

PLL division factor by 5

LL_RCC_PLLM_DIV_6

PLL division factor by 6

LL_RCC_PLLM_DIV_7

PLL division factor by 7

LL_RCC_PLLM_DIV_8

PLL division factor by 8

LL_RCC_PLLM_DIV_9

PLL division factor by 9

LL_RCC_PLLM_DIV_10

PLL division factor by 10

LL_RCC_PLLM_DIV_11

PLL division factor by 11

LL_RCC_PLLM_DIV_12

PLL division factor by 12

LL_RCC_PLLM_DIV_13

PLL division factor by 13

LL_RCC_PLLM_DIV_14

PLL division factor by 14

LL_RCC_PLLM_DIV_15

PLL division factor by 15

LL_RCC_PLLM_DIV_16

PLL division factor by 16

PLL division factor (PLLP)

LL_RCC_PLLP_DIV_2

Main PLL division factor for PLLP output by 2

LL_RCC_PLLP_DIV_3

Main PLL division factor for PLLP output by 3

LL_RCC_PLLP_DIV_4

Main PLL division factor for PLLP output by 4

LL_RCC_PLLP_DIV_5

Main PLL division factor for PLLP output by 5

LL_RCC_PLLP_DIV_6

Main PLL division factor for PLLP output by 6

LL_RCC_PLLP_DIV_7

Main PLL division factor for PLLP output by 7

LL_RCC_PLLP_DIV_8

Main PLL division factor for PLLP output by 8

LL_RCC_PLLP_DIV_9

Main PLL division factor for PLLP output by 9

LL_RCC_PLLP_DIV_10

Main PLL division factor for PLLP output by 10

LL_RCC_PLLP_DIV_11

Main PLL division factor for PLLP output by 11

LL_RCC_PLLP_DIV_12

Main PLL division factor for PLLP output by 12

LL_RCC_PLLP_DIV_13

Main PLL division factor for PLLP output by 13

LL_RCC_PLLP_DIV_14

Main PLL division factor for PLLP output by 14

LL_RCC_PLLP_DIV_15

Main PLL division factor for PLLP output by 15

LL_RCC_PLLP_DIV_16

Main PLL division factor for PLLP output by 16

LL_RCC_PLLP_DIV_17

Main PLL division factor for PLLP output by 17

LL_RCC_PLLP_DIV_18

Main PLL division factor for PLLP output by 18

LL_RCC_PLLP_DIV_19

Main PLL division factor for PLLP output by 19

LL_RCC_PLLP_DIV_20

Main PLL division factor for PLLP output by 20

LL_RCC_PLLP_DIV_21

Main PLL division factor for PLLP output by 21

LL_RCC_PLLP_DIV_22

Main PLL division factor for PLLP output by 22

LL_RCC_PLLP_DIV_23

Main PLL division factor for PLLP output by 23

LL_RCC_PLLP_DIV_24

Main PLL division factor for PLLP output by 24

LL_RCC_PLLP_DIV_25

Main PLL division factor for PLLP output by 25

LL_RCC_PLLP_DIV_26

Main PLL division factor for PLLP output by 26

LL_RCC_PLLP_DIV_27

Main PLL division factor for PLLP output by 27

LL_RCC_PLLP_DIV_28

Main PLL division factor for PLLP output by 28

LL_RCC_PLLP_DIV_29

Main PLL division factor for PLLP output by 29

LL_RCC_PLLP_DIV_30

Main PLL division factor for PLLP output by 30

LL_RCC_PLLP_DIV_31

Main PLL division factor for PLLP output by 31

PLL division factor (PLLQ)

LL_RCC_PLLQ_DIV_2

Main PLL division factor for PLLQ output by 2

LL_RCC_PLLQ_DIV_4

Main PLL division factor for PLLQ output by 4

LL_RCC_PLLQ_DIV_6

Main PLL division factor for PLLQ output by 6

LL_RCC_PLLQ_DIV_8

Main PLL division factor for PLLQ output by 8

PLL division factor (PLLR)

LL_RCC_PLLR_DIV_2

Main PLL division factor for PLLCLK (system clock) by 2

LL_RCC_PLLR_DIV_4

Main PLL division factor for PLLCLK (system clock) by 4

LL_RCC_PLLR_DIV_6

Main PLL division factor for PLLCLK (system clock) by 6

LL_RCC_PLLR_DIV_8

Main PLL division factor for PLLCLK (system clock) by 8

PLL entry clock source

LL_RCC_PLLSOURCE_NONE

No clock

LL_RCC_PLLSOURCE_HSI

HSI16 clock selected as PLL entry clock source

LL_RCC_PLLSOURCE_HSE

HSE clock selected as PLL entry clock source

Peripheral QUADSPI get clock source

LL_RCC_QUADSPI_CLKSOURCE_SYSCLK

SYSCLK used as QuadSPI clock source

LL_RCC_QUADSPI_CLKSOURCE_HSI

HSI used as QuadSPI clock source

LL_RCC_QUADSPI_CLKSOURCE_PLL

PLL used as QuadSPI clock source

LL_RCC_QUADSPI_CLKSOURCE

QuadSPI Clock source selection

Peripheral RNG get clock source

LL_RCC_RNG_CLKSOURCE

RNG Clock source selection

Peripheral RNG clock source selection

LL_RCC_RNG_CLKSOURCE_HSI48

HSI48 clock used as RNG clock source

LL_RCC_RNG_CLKSOURCE_PLL

PLL clock used as RNG clock source

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE

No clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_LSE

LSE oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_LSI

LSI oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_HSE_DIV32

HSE oscillator clock divided by 32 used as RTC clock

Peripheral SAI get clock source

LL_RCC_SAI1_CLKSOURCE

SAI1 Clock source selection

Peripheral SAI clock source selection

LL_RCC_SAI1_CLKSOURCE_SYSCLK

System clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PLL

PLL clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PIN

EXT clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_HSI

HSI clock used as SAI1 clock source

AHB prescaler

LL_RCC_SYSCLK_DIV_1

SYSCLK not divided

LL_RCC_SYSCLK_DIV_2

SYSCLK divided by 2

LL_RCC_SYSCLK_DIV_4

SYSCLK divided by 4

LL_RCC_SYSCLK_DIV_8

SYSCLK divided by 8

LL_RCC_SYSCLK_DIV_16

SYSCLK divided by 16

LL_RCC_SYSCLK_DIV_64

SYSCLK divided by 64

LL_RCC_SYSCLK_DIV_128

SYSCLK divided by 128

LL_RCC_SYSCLK_DIV_256

SYSCLK divided by 256

LL_RCC_SYSCLK_DIV_512

SYSCLK divided by 512

System clock switch

LL_RCC_SYS_CLKSOURCE_HSI

HSI selection as system clock

LL_RCC_SYS_CLKSOURCE_HSE

HSE selection as system clock

LL_RCC_SYS_CLKSOURCE_PLL

PLL selection as system clock

System clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_HSI

HSI used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_HSE

HSE used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_PLL

PLL used as system clock

Peripheral UART get clock source

LL_RCC_UART4_CLKSOURCE

UART4 Clock source selection

LL_RCC_UART5_CLKSOURCE

UART5 Clock source selection

Peripheral UART clock source selection

LL_RCC_UART4_CLKSOURCE_PCLK1

PCLK1 clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_SYSCLK

SYSCLK clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_HSI

HSI clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_LSE

LSE clock used as UART4 clock source

LL_RCC_UART5_CLKSOURCE_PCLK1

PCLK1 clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_SYSCLK

SYSCLK clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_HSI

HSI clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_LSE

LSE clock used as UART5 clock source

Peripheral USART get clock source

LL_RCC_USART1_CLKSOURCE

USART1 Clock source selection

LL_RCC_USART2_CLKSOURCE

USART2 Clock source selection

LL_RCC_USART3_CLKSOURCE

USART3 Clock source selection

Peripheral USART clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK2

PCLK2 clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_SYSCLK

SYSCLK clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_HSI

HSI clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_LSE

LSE clock used as USART1 clock source

LL_RCC_USART2_CLKSOURCE_PCLK1

PCLK1 clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_SYSCLK

SYSCLK clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_HSI

HSI clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_LSE

LSE clock used as USART2 clock source

LL_RCC_USART3_CLKSOURCE_PCLK1

PCLK1 clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_SYSCLK

SYSCLK clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_HSI

HSI clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_LSE

LSE clock used as USART3 clock source

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE

USB Clock source selection

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_HSI48

HSI48 clock used as USB clock source

LL_RCC_USB_CLKSOURCE_PLL

PLL clock used as USB clock source

Calculate frequencies

__LL_RCC_CALC_PLLCLK_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency on system domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE/HSI)
- __PLLM__: This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16
- __PLLN__: Between Min_Data = 8 and Max_Data = 127
- __PLLR__: This parameter can be one of the following values:
  - LL_RCC_PLLR_DIV_2
  - LL_RCC_PLLR_DIV_4
  - LL_RCC_PLLR_DIV_6
  - LL_RCC_PLLR_DIV_8

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_FREQ (HSE_VALUE,LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetR ());

__LL_RCC_CALC_PLLCLK_ADC_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency used on ADC domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE/HSI)
- __PLLM__: This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16
- __PLLN__: Between Min_Data = 8 and Max_Data = 127
- __PLLP__: This parameter can be one of the following values:
  - LL_RCC_PLLP_DIV_2
  - LL_RCC_PLLP_DIV_3
  - LL_RCC_PLLP_DIV_4
  - LL_RCC_PLLP_DIV_5
  - LL_RCC_PLLP_DIV_6
  - LL_RCC_PLLP_DIV_7
  - LL_RCC_PLLP_DIV_8
  - LL_RCC_PLLP_DIV_9
  - LL_RCC_PLLP_DIV_10
  - LL_RCC_PLLP_DIV_11
  - LL_RCC_PLLP_DIV_12
  - LL_RCC_PLLP_DIV_13
  - LL_RCC_PLLP_DIV_14
  - LL_RCC_PLLP_DIV_15
  - LL_RCC_PLLP_DIV_16
  - LL_RCC_PLLP_DIV_17
  - LL_RCC_PLLP_DIV_18
  - LL_RCC_PLLP_DIV_19
  - LL_RCC_PLLP_DIV_20
  - LL_RCC_PLLP_DIV_21
  - LL_RCC_PLLP_DIV_22
  - LL_RCC_PLLP_DIV_23
  - LL_RCC_PLLP_DIV_24
  - LL_RCC_PLLP_DIV_25
  - LL_RCC_PLLP_DIV_26
  - LL_RCC_PLLP_DIV_27
  - LL_RCC_PLLP_DIV_28
  - LL_RCC_PLLP_DIV_29
  - LL_RCC_PLLP_DIV_30
  - LL_RCC_PLLP_DIV_31

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_ADC_FREQ (HSE_VALUE,LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetP ());

__LL_RCC_CALC_PLLCLK_48M_FREQ

Description:

- Helper macro to calculate the PLLCLK frequency used on 48M domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSE/HSI)
- __PLLM__: This parameter can be one of the following values:
  - LL_RCC_PLLM_DIV_1
  - LL_RCC_PLLM_DIV_2
  - LL_RCC_PLLM_DIV_3
  - LL_RCC_PLLM_DIV_4
  - LL_RCC_PLLM_DIV_5
  - LL_RCC_PLLM_DIV_6
  - LL_RCC_PLLM_DIV_7
  - LL_RCC_PLLM_DIV_8
  - LL_RCC_PLLM_DIV_9
  - LL_RCC_PLLM_DIV_10
  - LL_RCC_PLLM_DIV_11
  - LL_RCC_PLLM_DIV_12
  - LL_RCC_PLLM_DIV_13
  - LL_RCC_PLLM_DIV_14
  - LL_RCC_PLLM_DIV_15
  - LL_RCC_PLLM_DIV_16
- __PLLN__: Between Min_Data = 8 and Max_Data = 127
- __PLLQ__: This parameter can be one of the following values:
  - LL_RCC_PLLQ_DIV_2
  - LL_RCC_PLLQ_DIV_4
  - LL_RCC_PLLQ_DIV_6
  - LL_RCC_PLLQ_DIV_8

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLLCLK_48M_FREQ (HSE_VALUE,LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetQ ());

__LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency (based on HSE/HSI/PLLCLK)
- __AHBPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

__LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

86 LL RNG Generic Driver

86.1 RNG Firmware driver registers structures

86.1.1 LL_RNG_InitTypeDef

LL_RNG_InitTypeDef is defined in the stm32g4xx_ll_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t LL_RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection. This parameter can be one value of ***RNG_LL_CED***.This parameter can be modified using unitary functions **LL_RNG_EnableClkErrorDetect()**.

86.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

86.2.1 Detailed description of functions

LL_RNG_Enable

Function name

__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name

__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabled (RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_IsEnabled

LL_RNG_EnableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_EnableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Enable Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_EnableClkErrorDetect

LL_RNG_DisableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_DisableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Disable RNG Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_DisableClkErrorDetect

LL_RNG_IsEnabledClkErrorDetect

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Check if RNG Clock Error Detection is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CED LL_RNG_IsEnabledClkErrorDetect

LL_RNG_IsActiveFlag_DRDY

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (RNG_TypeDef * RNGx)

Function description

Indicate if the RNG Data ready Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_IsActiveFlag_SEIS

LL_RNG_ClearFlag_CEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Clear Clock Error interrupt Status (CEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Clear Seed Error interrupt Status (SEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name

__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name

__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_DisableIT

LL_RNG_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name

__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (RNG_TypeDef * RNGx)

Function description

Return32-bit Random Number value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Generated:** 32-bit random value

Reference Manual to LL API cross reference:

- DR RNDATA LL_RNG_ReadRandData32

LL_RNG_Init

Function name

ErrorStatus LL_RNG_Init (RNG_TypeDef * RNGx, LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Initialize RNG registers according to the specified parameters in RNG_InitStruct.

Parameters

- **RNGx:** RNG Instance
- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure that contains the configuration information for the specified RNG peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are initialized according to RNG_InitStruct content
  - ERROR: not applicable

LL_RNG_StructInit

Function name

void LL_RNG_StructInit (LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Set each LL_RNG_InitTypeDef field to default value.

Parameters

- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_RNG_DeInit

Function name

ErrorStatus LL_RNG_DeInit (RNG_TypeDef * RNGx)

Function description

De-initialize RNG registers (Registers restored to their default values).

Parameters

- **RNGx:** RNG Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are de-initialized
  - ERROR: not applicable

86.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

86.3.1 RNG

RNG

Clock Error Detection

LL_RNG_CED_ENABLE

Clock error detection enabled

LL_RNG_CED_DISABLE

Clock error detection disabled

Get Flags Defines

LL_RNG_SR_DRDY

Register contains valid random data

LL_RNG_SR_CECS

Clock error current status

LL_RNG_SR_SECS

Seed error current status

LL_RNG_SR_CEIS

Clock error interrupt status

LL_RNG_SR_SEIS

Seed error interrupt status

IT Defines

LL_RNG_CR_IE

RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg

Description:

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg

Description:

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

87 LL RTC Generic Driver

87.1 RTC Firmware driver registers structures

87.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32g4xx_ll_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrescaler***
- ***uint32_t SynchPrescaler***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::HourFormat***

  Specifies the RTC Hours Format. This parameter can be a value of ***RTC_LL_EC_HOURFORMAT***This feature can be modified afterwards using unitary function **LL_RTC_SetHourFormat()**.
- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***

  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***

  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

87.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32g4xx_ll_rtc.h

Data Fields

- ***uint32_t TimeFormat***
- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint32_t LL_RTC_TimeTypeDef::TimeFormat***

  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_LL_EC_TIME_FORMAT***This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetFormat()**.
- ***uint8_t LL_RTC_TimeTypeDef::Hours***

  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the **LL_RTC_TIME_FORMAT_PM** is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the **LL_RTC_TIME_FORMAT_AM_OR_24** is selected.This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetHour()**.
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***

  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***

  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

87.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32g4xx_ll_rtc.h

Data Fields

- ***uint8_t WeekDay***

- ***uint8_t Month***
- ***uint8_t Day***
- ***uint8_t Year***

Field Documentation

- ***uint8_t LL_RTC_DateTypeDef::WeekDay***

  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_LL_EC_WEEKDAY***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetWeekDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Month***

  Specifies the RTC Date Month. This parameter can be a value of ***RTC_LL_EC_MONTH***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetMonth()**.
- ***uint8_t LL_RTC_DateTypeDef::Day***

  Specifies the RTC Date Day. This parameter must be a number between Min_Data = 1 and Max_Data = 31This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Year***

  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

87.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32g4xx_ll_rtc.h

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***

  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***

  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***

  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM BThis feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B
- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***

  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B.If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

87.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

87.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name

__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)

Function description

Set Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance
- **HourFormat:** This parameter can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (RTC_TypeDef * RTCx)

Function description

Get Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)

Function description

Select the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance
- **AlarmOutput:** This parameter can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (RTC_TypeDef * RTCx)

Function description

Get the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)

Function description

Set RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance
- **Output:** This parameter can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (RTC_TypeDef * RTCx)

Function description

Get RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_SetAlarmOutputType

LL_RTC_EnableInitMode

Function name

__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)

Function description

Enable initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name

__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)

Function description

Disable initialization mode (Free running mode)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_DisableInitMode

LL_RTC_SetOutputPolarity

Function name

__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)

Function description

Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is asserted)

Parameters

- **RTCx:** RTC Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (RTC_TypeDef * RTCx)

Function description

Get Output polarity.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Enable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Disable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (RTC_TypeDef * RTCx)

Function description

Check if Shadow registers bypass is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name

__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)

Function description

Enable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name

__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)

Function description

Disable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)

Function description

Set Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)

Function description

Set Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (RTC_TypeDef * RTCx)

Function description

Get Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7F

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (RTC_TypeDef * RTCx)

Function description

Get Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7FFF

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_GetSynchPrescaler

LL_RTC_EnableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)

Function description

Enable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)

Function description

Disable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_DisableWriteProtection

LL_RTC_EnableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_EnableTamperOutput (RTC_TypeDef * RTCx)

Function description

Enable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- When the tamper output is enabled, all external and internal tamper flags are ORed and routed to the TAMPALRM output.

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_EnableTamperOutput

LL_RTC_DisableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_DisableTamperOutput (RTC_TypeDef * RTCx)

Function description

Disable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_DisableTamperOutput

LL_RTC_IsTamperOutputEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsTamperOutputEnabled (RTC_TypeDef * RTCx)

Function description

Check if tamper output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_IsTamperOutputEnabled

LL_RTC_EnableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_EnableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Enable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_DisableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_DisableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Disable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_IsAlarmPullUpEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsAlarmPullUpEnabled (RTC_TypeDef * RTCx)

Function description

Check if internal pull-up in output mode is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_IsAlarmPullUpEnabled

LL_RTC_EnableOutput2

Function name

__STATIC_INLINE void LL_RTC_EnableOutput2 (RTC_TypeDef * RTCx)

Function description

Enable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- RTC_OUT2 mapping depends on both OSEL (LL_RTC_SetAlarmOutEvent) and COE (LL_RTC_CAL_SetOutputFreq) settings.
- RTC_OUT2 is not available ins VBAT mode.

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_EnableOutput2

LL_RTC_DisableOutput2

Function name

__STATIC_INLINE void LL_RTC_DisableOutput2 (RTC_TypeDef * RTCx)

Function description

Disable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_DisableOutput2

LL_RTC_IsOutput2Enabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsOutput2Enabled (RTC_TypeDef * RTCx)

Function description

Check if RTC_OUT2 output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_IsOutput2Enabled

LL_RTC_TIME_SetFormat

Function name

__STATIC_INLINE void LL_RTC_TIME_SetFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (RTC_TypeDef * RTCx)

Function description

Get time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_GetFormat

LL_RTC_TIME_SetHour

Function name

__STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_SetHour
- RTC_TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (RTC_TypeDef * RTCx)

Function description

Get Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_GetHour
- RTC_TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name

__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_SetMinute
- RTC_TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (RTC_TypeDef * RTCx)

Function description

Get Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_GetMinute
- RTC_TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name

__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_SetSecond
- RTC_TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (RTC_TypeDef * RTCx)

Function description

Get Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_GetSecond
- RTC_TR SU LL_RTC_TIME_GetSecond

LL_RTC_TIME_Config

Function name

__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- TimeFormat and Hours should follow the same format

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_Config
- RTC_TR HT LL_RTC_TIME_Config
- RTC_TR HU LL_RTC_TIME_Config
- RTC_TR MNT LL_RTC_TIME_Config
- RTC_TR MNU LL_RTC_TIME_Config
- RTC_TR ST LL_RTC_TIME_Config
- RTC_TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_Get (RTC_TypeDef * RTCx)

Function description

Get time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds (Format: 0x00HHMMSS).

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_Get
- RTC_TR HU LL_RTC_TIME_Get
- RTC_TR MNT LL_RTC_TIME_Get
- RTC_TR MNU LL_RTC_TIME_Get
- RTC_TR ST LL_RTC_TIME_Get
- RTC_TR SU LL_RTC_TIME_Get

LL_RTC_TIME_EnableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)

Function description

Memorize whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_EnableDayLightStore

LL_RTC_TIME_DisableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)

Function description

Disable memorization whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (RTC_TypeDef * RTCx)

Function description

Check if RTC Day Light Saving stored operation has been enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name

__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)

Function description

Subtract 1 hour (winter time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name

__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)

Function description

Add 1 hour (summer time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Sub second value in the synchronous prescaler counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **Sub:** second value (number between 0 and 65535)

Notes

- You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.

Reference Manual to LL API cross reference:

- RTC_SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name

__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)

Function description

Synchronize to a remote clock with a high degree of precision.

Parameters

- **RTCx:** RTC Instance
- **ShiftSecond:** This parameter can be one of the following values:
  - LL_RTC_SHIFT_SECOND_DELAY
  - LL_RTC_SHIFT_SECOND_ADVANCE
- **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)

Return values

- **None:**

Notes

- This operation effectively subtracts from (delays) or advance the clock of a fraction of a second.
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- When REFCKON is set, firmware must not write to Shift control register.

Reference Manual to LL API cross reference:

- RTC_SHIFTR ADD1S LL_RTC_TIME_Synchronize
- RTC_SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name

__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)

Function description

Set Year in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_SetYear
- RTC_DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (RTC_TypeDef * RTCx)

Function description

Get Year in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x99

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_GetYear
- RTC_DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set Week day.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_GetWeekDay

LL_RTC_DATE_SetMonth

Function name

__STATIC_INLINE void LL_RTC_DATE_SetMonth (RTC_TypeDef * RTCx, uint32_t Month)

Function description

Set Month in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_SetMonth
- RTC_DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (RTC_TypeDef * RTCx)

Function description

Get Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_GetMonth
- RTC_DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_SetDay
- RTC_DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (RTC_TypeDef * RTCx)

Function description

Get Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_GetDay
- RTC_DR DU LL_RTC_DATE_GetDay

LL_RTC_DATE_Config

Function name

__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)

Function description

Set date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Config
- RTC_DR MT LL_RTC_DATE_Config
- RTC_DR MU LL_RTC_DATE_Config
- RTC_DR DT LL_RTC_DATE_Config
- RTC_DR DU LL_RTC_DATE_Config
- RTC_DR YT LL_RTC_DATE_Config
- RTC_DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_Get (RTC_TypeDef * RTCx)

Function description

Get date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Get
- RTC_DR MT LL_RTC_DATE_Get
- RTC_DR MU LL_RTC_DATE_Get
- RTC_DR DT LL_RTC_DATE_Get
- RTC_DR DU LL_RTC_DATE_Get
- RTC_DR YT LL_RTC_DATE_Get
- RTC_DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm A masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (RTC_TypeDef * RTCx)

Function description

Get the Alarm A masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmA Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmA Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_SetDay
- RTC_ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (RTC_TypeDef * RTCx)

Function description

Get ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_GetDay
- RTC_ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set Alarm A time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get Alarm A time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_SetHour
- RTC_ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (RTC_TypeDef * RTCx)

Function description

Get ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetHour
- RTC_ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_SetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (RTC_TypeDef * RTCx)

Function description

Get ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_GetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_SetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (RTC_TypeDef * RTCx)

Function description

Get ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_GetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm A Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR ST LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetTime
- RTC_ALRMAR HU LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNT LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNU LL_RTC_ALMA_GetTime
- RTC_ALRMAR ST LL_RTC_ALMA_GetTime
- RTC_ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** Value between Min_Data=0x00 and Max_Data=0xF

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (RTC_TypeDef * RTCx)

Function description

Get Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xF

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm B masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (RTC_TypeDef * RTCx)

Function description

Get the Alarm B masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmB Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmB Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_SetDay
- RTC_ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (RTC_TypeDef * RTCx)

Function description

Get ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_GetDay
- RTC_ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set ALARM B time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get ALARM B time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_SetHour
- RTC_ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (RTC_TypeDef * RTCx)

Function description

Get ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetHour
- RTC_ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_SetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (RTC_TypeDef * RTCx)

Function description

Get ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_GetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_SetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (RTC_TypeDef * RTCx)

Function description

Get ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_GetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR ST LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetTime
- RTC_ALRMBR HU LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNT LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNU LL_RTC_ALMB_GetTime
- RTC_ALRMBR ST LL_RTC_ALMB_GetTime
- RTC_ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** Value between Min_Data=0x00 and Max_Data=0xF

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (RTC_TypeDef * RTCx)

Function description

Get Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL_RTC_ALMB_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_EnableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_EnableInternalEvent (RTC_TypeDef * RTCx)

Function description

Enable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_EnableInternalEvent

LL_RTC_TS_DisableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_DisableInternalEvent (RTC_TypeDef * RTCx)

Function description

Disable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_DisableInternalEvent

LL_RTC_TS_Enable

Function name

__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)

Function description

Enable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name

__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)

Function description

Disable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_Disable

LL_RTC_TS_SetActiveEdge

Function name

__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)

Function description

Set Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance
- **Edge:** This parameter can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting

Reference Manual to LL API cross reference:

- RTC_CR ITSEDGE LL_RTC_TS_SetActiveEdge

LL_RTC_TS_GetActiveEdge

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (RTC_TypeDef * RTCx)

Function description

Get Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSEDGE LL_RTC_TS_GetActiveEdge

LL_RTC_TS_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get Timestamp AM/PM notation (AM or 24-hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TS_TIME_FORMAT_AM
  - LL_RTC_TS_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (RTC_TypeDef * RTCx)

Function description

Get Timestamp Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetHour
- RTC_TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (RTC_TypeDef * RTCx)

Function description

Get Timestamp Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR MNT LL_RTC_TS_GetMinute
- RTC_TSTR HU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (RTC_TypeDef * RTCx)

Function description

Get Timestamp Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR ST LL_RTC_TS_GetSecond
- RTC_TSTR HU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (RTC_TypeDef * RTCx)

Function description

Get Timestamp time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetTime
- RTC_TSTR HU LL_RTC_TS_GetTime
- RTC_TSTR MNT LL_RTC_TS_GetTime
- RTC_TSTR MNU LL_RTC_TS_GetTime
- RTC_TSTR ST LL_RTC_TS_GetTime
- RTC_TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get Timestamp Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (RTC_TypeDef * RTCx)

Function description

Get Timestamp Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR MT LL_RTC_TS_GetMonth
- RTC_TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (RTC_TypeDef * RTCx)

Function description

Get Timestamp Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR DT LL_RTC_TS_GetDay
- RTC_TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (RTC_TypeDef * RTCx)

Function description

Get Timestamp date (WeekDay, Day and Month) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of Weekday, Day and Month

Notes

- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetDate
- RTC_TSDR MT LL_RTC_TS_GetDate
- RTC_TSDR MU LL_RTC_TS_GetDate
- RTC_TSDR DT LL_RTC_TS_GetDate
- RTC_TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get time-stamp sub second value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_TSDR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)

Function description

Activate timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)

Function description

Disable timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Enable (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_1
  - LL_RTC_TAMPER_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Enable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Enable

LL_RTC_TAMPER_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Disable (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Clear TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_1
  - LL_RTC_TAMPER_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Disable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Enable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_MASK_TAMPER1
  - LL_RTC_TAMPER_MASK_TAMPER2

Return values

- **None:**

Notes

- Associated Tamper IT must not enabled when tamper mask is set.

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_EnableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Disable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_MASK_TAMPER1
  - LL_RTC_TAMPER_MASK_TAMPER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_DisableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_NOERASE_TAMPER1
  - LL_RTC_TAMPER_NOERASE_TAMPER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1NOERASE LL_RTC_TAMPER_EnableEraseBKP
- TAMP_CR2 TAMP2NOERASE... LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_NOERASE_TAMPER1
  - LL_RTC_TAMPER_NOERASE_TAMPER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP
- TAMP_CR2 TAMP2NOERASE... LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (RTC_TypeDef * RTCx)

Function description

Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (RTC_TypeDef * RTCx)

Function description

Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (RTC_TypeDef * RTCx, uint32_t Duration)

Function description

Set RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance
- **Duration:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (RTC_TypeDef * RTCx, uint32_t FilterCount)

Function description

Set RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance
- **FilterCount:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (RTC_TypeDef * RTCx, uint32_t SamplingFreq)

Function description

Set Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance
- **SamplingFreq:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (RTC_TypeDef * RTCx)

Function description

Get Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP1
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel
- TAMP_CR2 TAMP2TRG... LL_RTC_TAMPER_EnableActiveLevel

LL_RTC_TAMPER_DisableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP1
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel
- TAMP_CR2 TAMP2TRG... LL_RTC_TAMPER_DisableActiveLevel

LL_RTC_TAMPER_ITAMP_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Enable (RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Enable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ITAMP1
  - LL_RTC_TAMPER_ITAMP3
  - LL_RTC_TAMPER_ITAMP4
  - LL_RTC_TAMPER_ITAMP5
  - LL_RTC_TAMPER_ITAMP6

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP3E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP4E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP5E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP6E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP7E... LL_RTC_TAMPER_ITAMP_Enable

LL_RTC_TAMPER_ITAMP_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Disable (RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Disable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ITAMP1
  - LL_RTC_TAMPER_ITAMP3
  - LL_RTC_TAMPER_ITAMP4
  - LL_RTC_TAMPER_ITAMP5
  - LL_RTC_TAMPER_ITAMP6

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP3E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP4E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP5E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP6E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP7E... LL_RTC_TAMPER_ITAMP_Disable

LL_RTC_WAKEUP_Enable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)

Function description

Select Wakeup clock.

Parameters

- **RTCx:** RTC Instance
- **WakeupClock:** This parameter can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ICSR WUTWF bit = 1

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (RTC_TypeDef * RTCx)

Function description

Get Wakeup clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)

Function description

Set Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance
- **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Bit can be written only when WUTWF is set to 1 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (RTC_TypeDef * RTCx)

Function description

Get Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BKP_SetRegister

Function name

__STATIC_INLINE void LL_RTC_BKP_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified Backup data register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3 ...
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_SetRegister

LL_RTC_BKP_GetRegister

Function name

__STATIC_INLINE uint32_t LL_RTC_BKP_GetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3 ...

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name

__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)

Function description

Set Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance
- **Frequency:** This parameter can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Return values

- **None:**

Notes

- Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_SetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (RTC_TypeDef * RTCx)

Function description

Get Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_GetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)

Function description

Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance
- **Pulse:** This parameter can be one of the following values:
  - LL_RTC_CALIB_INSERTPULSE_NONE
  - LL_RTC_CALIB_INSERTPULSE_SET

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (RTC_TypeDef * RTCx)

Function description

Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)

Function description

Set the calibration cycle period.

Parameters

- **RTCx:** RTC Instance
- **Period:** This parameter can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_SetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (RTC_TypeDef * RTCx)

Function description

Get the calibration cycle period.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_GetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_GetPeriod

LL_RTC_CAL_SetMinus

Function name

__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)

Function description

Set Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (RTC_TypeDef * RTCx)

Function description

Get Calibration minus.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data= 0x1FF

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_IsActiveFlag_ITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITS (RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ITSF LL_RTC_IsActiveFlag_ITS

LL_RTC_IsActiveFlag_RECALP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (RTC_TypeDef * RTCx)

Function description

Get Recalibration pending Flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TSOV

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (RTC_TypeDef * RTCx)

Function description

Get Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Get Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Get Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_ClearFlag_ITS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITS (RTC_TypeDef * RTCx)

Function description

Clear Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CITSF LL_RTC_ClearFlag_ITS

LL_RTC_ClearFlag_TSOV

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)

Function description

Clear Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CWUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Clear Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Clear Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_IsActiveFlag_INIT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (RTC_TypeDef * RTCx)

Function description

Get Initialization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)

Function description

Get Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)

Function description

Clear Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (RTC_TypeDef * RTCx)

Function description

Get Initialization status flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (RTC_TypeDef * RTCx)

Function description

Get Shift operation pending flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRBW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBW (RTC_TypeDef * RTCx)

Function description

Get Alarm B write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR ALRBWF LL_RTC_IsActiveFlag_ALRBW

LL_RTC_IsActiveFlag_ALRAW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (RTC_TypeDef * RTCx)

Function description

Get Alarm A write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR ALRAWF LL_RTC_IsActiveFlag_ALRAW

LL_RTC_IsActiveFlag_ALRAM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAM (RTC_TypeDef * RTCx)

Function description

Get Alarm A masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRAMF LL_RTC_IsActiveFlag_ALRAM

LL_RTC_IsActiveFlag_ALRBM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBM (RTC_TypeDef * RTCx)

Function description

Get Alarm B masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRBMF LL_RTC_IsActiveFlag_ALRBM

LL_RTC_IsActiveFlag_WUTM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTM (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR WUTMF LL_RTC_IsActiveFlag_WUTM

LL_RTC_IsActiveFlag_TSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSM (RTC_TypeDef * RTCx)

Function description

Get Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSMF LL_RTC_IsActiveFlag_TSM

LL_RTC_IsActiveFlag_TSOVM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOVM (RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSOVMF LL_RTC_IsActiveFlag_TSOVM

LL_RTC_IsActiveFlag_ITSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITSM (RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ITSMF LL_RTC_IsActiveFlag_ITSM

LL_RTC_IsActiveFlag_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (RTC_TypeDef * RTCx)

Function description

Get tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (RTC_TypeDef * RTCx)

Function description

Get tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (RTC_TypeDef * RTCx)

Function description

Get tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3 (RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP3F LL_RTC_IsActiveFlag_ITAMP3

LL_RTC_IsActiveFlag_ITAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP4 (RTC_TypeDef * RTCx)

Function description

Get internal tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP4F LL_RTC_IsActiveFlag_ITAMP4

LL_RTC_IsActiveFlag_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5 (RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP5F LL_RTC_IsActiveFlag_ITAMP5

LL_RTC_IsActiveFlag_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6 (RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP6F LL_RTC_IsActiveFlag_ITAMP6

LL_RTC_IsActiveFlag_TAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1M (RTC_TypeDef * RTCx)

Function description

Get tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP1MF LL_RTC_IsActiveFlag_TAMP1M

LL_RTC_IsActiveFlag_TAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2M (RTC_TypeDef * RTCx)

Function description

Get tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP2MF LL_RTC_IsActiveFlag_TAMP2M

LL_RTC_IsActiveFlag_TAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3M (RTC_TypeDef * RTCx)

Function description

Get tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP3MF LL_RTC_IsActiveFlag_TAMP3M

LL_RTC_IsActiveFlag_ITAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3M (RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP3MF LL_RTC_IsActiveFlag_ITAMP3M

LL_RTC_IsActiveFlag_ITAMP4M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP4M (RTC_TypeDef * RTCx)

Function description

Get internal tamper 4 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP4MF LL_RTC_IsActiveFlag_ITAMP4M

LL_RTC_IsActiveFlag_ITAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5M (RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP5MF LL_RTC_IsActiveFlag_ITAMP5M

LL_RTC_IsActiveFlag_ITAMP6M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6M (RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP6MF LL_RTC_IsActiveFlag_ITAMP6M

LL_RTC_ClearFlag_TAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (RTC_TypeDef * RTCx)

Function description

Clear tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)

Function description

Clear tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (RTC_TypeDef * RTCx)

Function description

Clear tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP3 (RTC_TypeDef * RTCx)

Function description

Clear internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP3F LL_RTC_ClearFlag_ITAMP3

LL_RTC_ClearFlag_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP4 (RTC_TypeDef * RTCx)

Function description

Clear internal tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP4F LL_RTC_ClearFlag_ITAMP4

LL_RTC_ClearFlag_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP5 (RTC_TypeDef * RTCx)

Function description

Clear internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP5F LL_RTC_ClearFlag_ITAMP5

LL_RTC_ClearFlag_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP6 (RTC_TypeDef * RTCx)

Function description

Clear internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP6F LL_RTC_ClearFlag_ITAMP6

LL_RTC_EnableIT_TS

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)

Function description

Enable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)

Function description

Disable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR LL_RTC_DisableIT_WUT

LL_RTC_EnableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Enable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Disable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Enable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Disable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_IsEnabledIT_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (RTC_TypeDef * RTCx)

Function description

Check if Time-stamp interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_IsEnabledIT_TS

LL_RTC_IsEnabledIT_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (RTC_TypeDef * RTCx)

Function description

Check if Alarm B interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_IsEnabledIT_ALRB

LL_RTC_IsEnabledIT_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (RTC_TypeDef * RTCx)

Function description

Check if Alarm A interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_IsEnabledIT_ALRA

LL_RTC_EnableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (RTC_TypeDef * RTCx)

Function description

Enable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (RTC_TypeDef * RTCx)

Function description

Disable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_DisableIT_TAMP1

LL_RTC_EnableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (RTC_TypeDef * RTCx)

Function description

Enable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (RTC_TypeDef * RTCx)

Function description

Disable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (RTC_TypeDef * RTCx)

Function description

Enable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (RTC_TypeDef * RTCx)

Function description

Disable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_DisableIT_TAMP3

LL_RTC_EnableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP3 (RTC_TypeDef * RTCx)

Function description

Enable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_EnableIT_ITAMP3

LL_RTC_DisableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP3 (RTC_TypeDef * RTCx)

Function description

Disable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_DisableIT_ITAMP3

LL_RTC_EnableIT_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP4 (RTC_TypeDef * RTCx)

Function description

Enable internal tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_EnableIT_ITAMP4

LL_RTC_DisableIT_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP4 (RTC_TypeDef * RTCx)

Function description

Disable internal tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_DisableIT_ITAMP4

LL_RTC_EnableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP5 (RTC_TypeDef * RTCx)

Function description

Enable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_EnableIT_ITAMP5

LL_RTC_DisableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP5 (RTC_TypeDef * RTCx)

Function description

Disable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_DisableIT_ITAMP5

LL_RTC_EnableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP6 (RTC_TypeDef * RTCx)

Function description

Enable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_EnableIT_ITAMP6

LL_RTC_DisableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP6 (RTC_TypeDef * RTCx)

Function description

Disable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_DisableIT_ITAMP6

LL_RTC_IsEnabledIT_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (RTC_TypeDef * RTCx)

Function description

Check if tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_IsEnabledIT_TAMP1

LL_RTC_IsEnabledIT_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (RTC_TypeDef * RTCx)

Function description

Check if tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (RTC_TypeDef * RTCx)

Function description

Check if tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP3 (RTC_TypeDef * RTCx)

Function description

Check if internal tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_IsEnabledIT_ITAMP3

LL_RTC_IsEnabledIT_ITAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP4 (RTC_TypeDef * RTCx)

Function description

Check if internal tamper 4 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_IsEnabledIT_ITAMP4

LL_RTC_IsEnabledIT_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP5 (RTC_TypeDef * RTCx)

Function description

Check if internal tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_IsEnabledIT_ITAMP5

LL_RTC_IsEnabledIT_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP6 (RTC_TypeDef * RTCx)

Function description

Check if internal tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_IsEnabledIT_ITAMP6

LL_RTC_DeInit

Function name

ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)

Function description

De-Initializes the RTC registers to their default reset values.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are de-initialized
  - ERROR: RTC registers are not de-initialized

Notes

- This function does not reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name

ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Initializes the RTC registers according to the specified parameters in RTC_InitStruct.

Parameters

- **RTCx:** RTC Instance
- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are initialized
  - ERROR: RTC registers are not initialized

Notes

- The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name

void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Set each LL_RTC_InitTypeDef field to default value.

Parameters

- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_TIME_Init

Function name

ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set the RTC current time.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Time register is configured
  - ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name

void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters

- **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_DATE_Init

Function name

ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set the RTC current date.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name

void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)

Parameters

- **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMA_Init

Function name

ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm A.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMA registers are configured
  - ERROR: ALARMA registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name

ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm B.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMB registers are configured
  - ERROR: ALARMB registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name

void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMB_StructInit

Function name

void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMB field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_EnterInitMode

Function name

ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)

Function description

Enters the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC is in Init mode
  - ERROR: RTC is not in Init mode

Notes

- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name

ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)

Function description

Exit the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC exited from in Init mode
  - ERROR: Not applicable

Notes

- When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name

ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)

Function description

Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are synchronised
  - ERROR: RTC registers are not synchronised

Notes

- The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

87.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

87.3.1 RTC

RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

LL_RTC_ALARMOUT_ALMB

Alarm B output enabled

LL_RTC_ALARMOUT_WAKEUP

Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM is open-drain output

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM is push-pull output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE

No masks applied on Alarm A

LL_RTC_ALMA_MASK_DATEWEEKDAY

Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS

Hours do not care in Alarm A comparison

LL_RTC_ALMA_MASK_MINUTES

Minutes do not care in Alarm A comparison

LL_RTC_ALMA_MASK_SECONDS

Seconds do not care in Alarm A comparison

LL_RTC_ALMA_MASK_ALL

Masks all

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMA_TIME_FORMAT_PM

PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE

Alarm A Date is selected

LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY

Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE

No masks applied on Alarm B

LL_RTC_ALMB_MASK_DATEWEEKDAY

Date/day do not care in Alarm B comparison

LL_RTC_ALMB_MASK_HOURS

Hours do not care in Alarm B comparison

LL_RTC_ALMB_MASK_MINUTES

Minutes do not care in Alarm B comparison

LL_RTC_ALMB_MASK_SECONDS

Seconds do not care in Alarm B comparison

LL_RTC_ALMB_MASK_ALL

Masks all

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMB_TIME_FORMAT_PM

PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE

Alarm B Date is selected

LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY

Alarm B WeekDay is selected

BACKUP

LL_RTC_BKP_NUMBER

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17

LL_RTC_BKP_DR18

LL_RTC_BKP_DR19

LL_RTC_BKP_DR20

LL_RTC_BKP_DR21

LL_RTC_BKP_DR22

LL_RTC_BKP_DR23

LL_RTC_BKP_DR24

LL_RTC_BKP_DR25

LL_RTC_BKP_DR26

LL_RTC_BKP_DR27

LL_RTC_BKP_DR28

LL_RTC_BKP_DR29

LL_RTC_BKP_DR30

LL_RTC_BKP_DR31

Calibration pulse insertion

LL_RTC_CALIB_INSERTPULSE_NONE

No RTCCLK pulses are added

LL_RTC_CALIB_INSERTPULSE_SET

One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration output

LL_RTC_CALIB_OUTPUT_NONE

Calibration output disabled

LL_RTC_CALIB_OUTPUT_1HZ

Calibration output is 1 Hz

LL_RTC_CALIB_OUTPUT_512HZ

Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC

Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_16SEC

Use a 16-second calibration cycle period

LL_RTC_CALIB_PERIOD_8SEC

Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN

Binary data format

LL_RTC_FORMAT_BCD

BCD data format

Get Flags Defines

LL_RTC_SCR_ITSF

LL_RTC_SCR_TSOVF

LL_RTC_SCR_TSF

LL_RTC_SCR_WUTF

LL_RTC_SCR_ALRBF

LL_RTC_CSR_ALRAF

LL_RTC_ICSR_RECALPF

LL_RTC_ICSR_INITF

LL_RTC_ICSR_RSF

LL_RTC_ICSR_INITS

LL_RTC_ICSR_SHPF

LL_RTC_ICSR_WUTWF

LL_RTC_ICSR_ALRBWF

LL_RTC_ICSR_ALRAWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR

24 hour/day format

LL_RTC_HOURFORMAT_AMPM

AM/PM hour format

INTERNAL TAMPER

LL_RTC_TAMPER_ITAMP1

Internal tamper 1: RTC supply voltage monitoring

LL_RTC_TAMPER_ITAMP3

Internal tamper 3: LSE monitoring

LL_RTC_TAMPER_ITAMP4

Internal tamper 4: HSE monitoring

LL_RTC_TAMPER_ITAMP5

Internal tamper 5: RTC calendar overflow

LL_RTC_TAMPER_ITAMP6

Internal tamper 6: Test mode entry

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

MONTH

LL_RTC_MONTH_JANUARY

January

LL_RTC_MONTH_FEBRUARY

February

LL_RTC_MONTH_MARCH

March

LL_RTC_MONTH_APRIL

April

LL_RTC_MONTH_MAY

May

LL_RTC_MONTH_JUNE

June

LL_RTC_MONTH_JULY

July

LL_RTC_MONTH_AUGUST

August

LL_RTC_MONTH_SEPTEMBER

September

LL_RTC_MONTH_OCTOBER

October

LL_RTC_MONTH_NOVEMBER

November

LL_RTC_MONTH_DECEMBER

December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH

Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW

Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

TAMPER

LL_RTC_TAMPER_1

Tamper 1 input detection

LL_RTC_TAMPER_2

Tamper 2 input detection

LL_RTC_TAMPER_3

Tamper 3 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1

Tamper 1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

Tamper 2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

Tamper 3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

LL_RTC_TAMPER_DURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

LL_RTC_TAMPER_DURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

LL_RTC_TAMPER_DURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE

Tamper filter is disabled

LL_RTC_TAMPER_FILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level.

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1

Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware.The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER2

Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

LL_RTC_TAMPER_MASK_TAMPER3

Tamper 3 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1

Tamper 1 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER2

Tamper 2 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER3

Tamper 3 event does not erase the backup registers.

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

LL_RTC_TAMPER_SAMPLFREQDIV_16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

LL_RTC_TAMPER_SAMPLFREQDIV_8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

LL_RTC_TAMPER_SAMPLFREQDIV_4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

LL_RTC_TAMPER_SAMPLFREQDIV_2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPLFREQDIV_1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

LL_RTC_TAMPER_SAMPLFREQDIV_512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

LL_RTC_TAMPER_SAMPLFREQDIV_256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING

RTC_TS input rising edge generates a time-stamp event

LL_RTC_TIMESTAMP_EDGE_FALLING

RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24

AM or 24-hour format

LL_RTC_TIME_FORMAT_PM

PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_TS_TIME_FORMAT_PM

PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16

RTC/16 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_8

RTC/8 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_4

RTC/4 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_2

RTC/2 clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE

ck_spre (usually 1 Hz) clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY

Monday

LL_RTC_WEEKDAY_TUESDAY

Tuesday

LL_RTC_WEEKDAY_WEDNESDAY

Wednesday

LL_RTC_WEEKDAY_THURSDAY

Thrusday

LL_RTC_WEEKDAY_FRIDAY

Friday

LL_RTC_WEEKDAY_SATURDAY

Saturday

LL_RTC_WEEKDAY_SUNDAY

Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

88 LL SPI Generic Driver

88.1 SPI Firmware driver registers structures

88.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32g4xx_ll_spi.h

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***

  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***

  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***

  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***

  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***

  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_SPI_InitTypeDef::NSS***

  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***

  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.

  Note:

  – The communication clock is derived from the master clock. The slave clock does not need to be set.

  This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.
- ***uint32_t LL_SPI_InitTypeDef::BitOrder***

  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.

- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***

  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***

  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

88.1.2 LL_I2S_InitTypeDef

LL_I2S_InitTypeDef is defined in the stm32g4xx_ll_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t ClockPolarity***

Field Documentation

- ***uint32_t LL_I2S_InitTypeDef::Mode***

  Specifies the I2S operating mode. This parameter can be a value of ***I2S_LL_EC_MODE***This feature can be modified afterwards using unitary function **LL_I2S_SetTransferMode()**.
- ***uint32_t LL_I2S_InitTypeDef::Standard***

  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_LL_EC_STANDARD***This feature can be modified afterwards using unitary function **LL_I2S_SetStandard()**.
- ***uint32_t LL_I2S_InitTypeDef::DataFormat***

  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_LL_EC_DATA_FORMAT***This feature can be modified afterwards using unitary function **LL_I2S_SetDataFormat()**.
- ***uint32_t LL_I2S_InitTypeDef::MCLKOutput***

  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_LL_EC_MCLK_OUTPUT***This feature can be modified afterwards using unitary functions **LL_I2S_EnableMasterClock()** or **LL_I2S_DisableMasterClock**.
- ***uint32_t LL_I2S_InitTypeDef::AudioFreq***

  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_LL_EC_AUDIO_FREQ***Audio Frequency can be modified afterwards using Reference manual formulas to calculate Prescaler Linear, Parity and unitary functions **LL_I2S_SetPrescalerLinear()** and **LL_I2S_SetPrescalerParity()** to set it.
- ***uint32_t LL_I2S_InitTypeDef::ClockPolarity***

  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_LL_EC_POLARITY***This feature can be modified afterwards using unitary function **LL_I2S_SetClockPolarity()**.

88.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

88.2.1 Detailed description of functions

LL_SPI_Enable

Function name

__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)

Function description

Enable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name

__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)

Function description

Disable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- When disabling the SPI, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)

Function description

Check if SPI peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_IsEnabled

LL_SPI_SetMode

Function name

__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set SPI operation mode to Master or Slave.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing.

Reference Manual to LL API cross reference:

- CR1 MSTR LL_SPI_SetMode
- CR1 SSI LL_SPI_SetMode

LL_SPI_GetMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMode (SPI_TypeDef * SPIx)

Function description

Get SPI operation mode (Master or Slave)

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Reference Manual to LL API cross reference:

- CR1 MSTR LL_SPI_GetMode
- CR1 SSI LL_SPI_GetMode

LL_SPI_SetStandard

Function name

__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set serial protocol used.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Return values

- **None:**

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR2 FRF LL_SPI_SetStandard

LL_SPI_GetStandard

Function name

__STATIC_INLINE uint32_t LL_SPI_GetStandard (SPI_TypeDef * SPIx)

Function description

Get serial protocol used.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Reference Manual to LL API cross reference:

- CR2 FRF LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name

__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)

Function description

Set clock phase.

Parameters

- **SPIx:** SPI Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (SPI_TypeDef * SPIx)

Function description

Get clock phase.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Reference Manual to LL API cross reference:

- CR1 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name

__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (SPI_TypeDef * SPIx)

Function description

Get clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CR1 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetBaudRatePrescaler

Function name

__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t BaudRate)

Function description

Set baud rate prescaler.

Parameters

- **SPIx:** SPI Instance
- **BaudRate:** This parameter can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Return values

- **None:**

Notes

- These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.

Reference Manual to LL API cross reference:

- CR1 BR LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (SPI_TypeDef * SPIx)

Function description

Get baud rate prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Reference Manual to LL API cross reference:

- CR1 BR LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set transfer bit order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 LSBFIRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (SPI_TypeDef * SPIx)

Function description

Get transfer bit order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Reference Manual to LL API cross reference:

- CR1 LSBFIRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name

__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)

Function description

Set transfer direction mode.

Parameters

- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- For Half-Duplex mode, Rx Direction is set by default. In master mode, the MOSI pin is used and in slave mode, the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 RXONLY LL_SPI_SetTransferDirection
- CR1 BIDIMODE LL_SPI_SetTransferDirection
- CR1 BIDIOE LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (SPI_TypeDef * SPIx)

Function description

Get transfer direction mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Reference Manual to LL API cross reference:

- CR1 RXONLY LL_SPI_GetTransferDirection
- CR1 BIDIMODE LL_SPI_GetTransferDirection
- CR1 BIDIOE LL_SPI_GetTransferDirection

LL_SPI_SetDataWidth

Function name

__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)

Function description

Set frame data width.

Parameters

- **SPIx:** SPI Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DS LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (SPI_TypeDef * SPIx)

Function description

Get frame data width.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT

Reference Manual to LL API cross reference:

- CR2 DS LL_SPI_GetDataWidth

LL_SPI_SetRxFIFOThreshold

Function name

__STATIC_INLINE void LL_SPI_SetRxFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of RXFIFO that triggers an RXNE event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_SPI_RX_FIFO_TH_HALF
  - LL_SPI_RX_FIFO_TH_QUARTER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 FRXTH LL_SPI_SetRxFIFOThreshold

LL_SPI_GetRxFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOThreshold (SPI_TypeDef * SPIx)

Function description

Get threshold of RXFIFO that triggers an RXNE event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_TH_HALF
  - LL_SPI_RX_FIFO_TH_QUARTER

Reference Manual to LL API cross reference:

- CR2 FRXTH LL_SPI_GetRxFIFOThreshold

LL_SPI_EnableCRC

Function name

__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name

__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)

Function description

Check if CRC is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name

__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)

Function description

Set CRC Length.

Parameters

- **SPIx:** SPI Instance
- **CRCLength:** This parameter can be one of the following values:
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_16BIT

Return values

- **None:**

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 CRCL LL_SPI_SetCRCWidth

LL_SPI_GetCRCWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (SPI_TypeDef * SPIx)

Function description

Get CRC Length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_16BIT

Reference Manual to LL API cross reference:

- CR1 CRCL LL_SPI_GetCRCWidth

LL_SPI_SetCRCNext

Function name

__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)

Function description

Set CRCNext to transfer CRC on the line.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit has to be written as soon as the last data is written in the SPIx_DR register.

Reference Manual to LL API cross reference:

- CR1 CRCNEXT LL_SPI_SetCRCNext

LL_SPI_SetCRCPolynomial

Function name

__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)

Function description

Set polynomial for CRC calculation.

Parameters

- **SPIx:** SPI Instance
- **CRCPoly:** This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRCPR CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (SPI_TypeDef * SPIx)

Function description

Get polynomial for CRC calculation.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- CRCPR CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_GetRxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)

Function description

Get Rx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)

Function description

Get Tx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_SetNSSMode

Function name

__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)

Function description

Set NSS mode.

Parameters

- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Return values

- **None:**

Notes

- LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 SSM LL_SPI_SetNSSMode
-
- CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)

Function description

Get NSS mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:

- CR1 SSM LL_SPI_GetNSSMode
-
- CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Enable NSS pulse management.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR2 NSSP LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Disable NSS pulse management.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR2 NSSP LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (SPI_TypeDef * SPIx)

Function description

Check if NSS pulse is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR2 NSSP LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)

Function description

Check if Rx buffer is not empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_SPI_IsActiveFlag_RXNE

LL_SPI_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE (SPI_TypeDef * SPIx)

Function description

Check if Tx buffer is empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXE LL_SPI_IsActiveFlag_TXE

LL_SPI_IsActiveFlag_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Get CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CRCERR LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (SPI_TypeDef * SPIx)

Function description

Get mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (SPI_TypeDef * SPIx)

Function description

Get overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_BSY

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_BSY (SPI_TypeDef * SPIx)

Function description

Get busy flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- The BSY flag is cleared under any one of the following conditions: -When the SPI is correctly disabled -When a fault is detected in Master mode (MODF bit set to 1) -In Master mode, when it finishes a data transmission and no new data is ready to be sent -In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.

Reference Manual to LL API cross reference:

- SR BSY LL_SPI_IsActiveFlag_BSY

LL_SPI_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (SPI_TypeDef * SPIx)

Function description

Get frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR FRE LL_SPI_IsActiveFlag_FRE

LL_SPI_GetRxFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOLevel (SPI_TypeDef * SPIx)

Function description

Get FIFO reception Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_EMPTY
  - LL_SPI_RX_FIFO_QUARTER_FULL
  - LL_SPI_RX_FIFO_HALF_FULL
  - LL_SPI_RX_FIFO_FULL

Reference Manual to LL API cross reference:

- SR FRLVL LL_SPI_GetRxFIFOLevel

LL_SPI_GetTxFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxFIFOLevel (SPI_TypeDef * SPIx)

Function description

Get FIFO Transmission Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_TX_FIFO_EMPTY
  - LL_SPI_TX_FIFO_QUARTER_FULL
  - LL_SPI_TX_FIFO_HALF_FULL
  - LL_SPI_TX_FIFO_FULL

Reference Manual to LL API cross reference:

- SR FTLVL LL_SPI_GetTxFIFOLevel

LL_SPI_ClearFlag_CRCERR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Clear CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CRCERR LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)

Function description

Clear mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Clearing this flag is done by a read access to the SPIx_SR register followed by a write access to the SPIx_CR1 register

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Clearing this flag is done by a read access to the SPIx_DR register followed by a read access to the SPIx_SR register

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Clearing this flag is done by reading SPIx_SR register

Reference Manual to LL API cross reference:

- SR FRE LL_SPI_ClearFlag_FRE

LL_SPI_EnableIT_ERR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef * SPIx)

Function description

Enable error interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_SPI_EnableIT_ERR

LL_SPI_EnableIT_RXNE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Enable Rx buffer not empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_SPI_EnableIT_RXNE

LL_SPI_EnableIT_TXE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef * SPIx)

Function description

Enable Tx buffer empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_SPI_EnableIT_TXE

LL_SPI_DisableIT_ERR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef * SPIx)

Function description

Disable error interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_SPI_DisableIT_ERR

LL_SPI_DisableIT_RXNE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Disable Rx buffer not empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_SPI_DisableIT_RXNE

LL_SPI_DisableIT_TXE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef * SPIx)

Function description

Disable Tx buffer empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_SPI_DisableIT_TXE

LL_SPI_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (SPI_TypeDef * SPIx)

Function description

Check if error interrupt is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_SPI_IsEnabledIT_ERR

LL_SPI_IsEnabledIT_RXNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)

Function description

Check if Rx buffer not empty interrupt is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_SPI_IsEnabledIT_RXNE

LL_SPI_IsEnabledIT_TXE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXE (SPI_TypeDef * SPIx)

Function description

Check if Tx buffer empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_SPI_IsEnabledIT_TXE

LL_SPI_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_SetDMAParity_RX

Function name

__STATIC_INLINE void LL_SPI_SetDMAParity_RX (SPI_TypeDef * SPIx, uint32_t Parity)

Function description

Set parity of Last DMA reception.

Parameters

- **SPIx:** SPI Instance
- **Parity:** This parameter can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 LDMARX LL_SPI_SetDMAParity_RX

LL_SPI_GetDMAParity_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_RX (SPI_TypeDef * SPIx)

Function description

Get parity configuration for Last DMA reception.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Reference Manual to LL API cross reference:

- CR2 LDMARX LL_SPI_GetDMAParity_RX

LL_SPI_SetDMAParity_TX

Function name

__STATIC_INLINE void LL_SPI_SetDMAParity_TX (SPI_TypeDef * SPIx, uint32_t Parity)

Function description

Set parity of Last DMA transmission.

Parameters

- **SPIx:** SPI Instance
- **Parity:** This parameter can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 LDMATX LL_SPI_SetDMAParity_TX

LL_SPI_GetDMAParity_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_TX (SPI_TypeDef * SPIx)

Function description

Get parity configuration for Last DMA transmission.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Reference Manual to LL API cross reference:

- CR2 LDMATX LL_SPI_GetDMAParity_TX

LL_SPI_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read 8-Bits in the data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read 16-Bits in the data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- DR DR LL_SPI_ReceiveData16

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write 8-Bits in the data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write 16-Bits in the data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_SPI_TransmitData16

LL_SPI_DeInit

Function name

ErrorStatus LL_SPI_DeInit (SPI_TypeDef * SPIx)

Function description

De-initialize the SPI registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name

ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Initialize the SPI registers according to the specified parameters in SPI_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name

void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_Enable

Function name

__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)

Function description

Select I2S mode and Enable I2S peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SMOD LL_I2S_Enable
- I2SCFGR I2SE LL_I2S_Enable

LL_I2S_Disable

Function name

__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)

Function description

Disable I2S peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SE LL_I2S_Disable

LL_I2S_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabled (SPI_TypeDef * SPIx)

Function description

Check if I2S peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SCFGR I2SE LL_I2S_IsEnabled

LL_I2S_SetDataFormat

Function name

__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataFormat)

Function description

Set I2S data frame length.

Parameters

- **SPIx:** SPI Instance
- **DataFormat:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name

__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (SPI_TypeDef * SPIx)

Function description

Get I2S data frame length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_GetDataFormat
- I2SCFGR CHLEN LL_I2S_GetDataFormat

LL_I2S_SetClockPolarity

Function name

__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set I2S clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (SPI_TypeDef * SPIx)

Function description

Get I2S clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name

__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S standard protocol.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name

__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)

Function description

Get I2S standard protocol.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name

__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set I2S transfer mode.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (SPI_TypeDef * SPIx)

Function description

Get I2S transfer mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_SetPrescalerLinear

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint8_t PrescalerLinear)

Function description

Set I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (SPI_TypeDef * SPIx)

Function description

Get I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- I2SPR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)

Function description

Set I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (SPI_TypeDef * SPIx)

Function description

Get I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SPR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name

__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)

Function description

Enable the master clock ouput (Pin MCK)

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name

__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)

Function description

Disable the master clock ouput (Pin MCK)

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (SPI_TypeDef * SPIx)

Function description

Check if the master clock ouput (Pin MCK) is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_EnableAsyncStart

Function name

__STATIC_INLINE void LL_I2S_EnableAsyncStart (SPI_TypeDef * SPIx)

Function description

Enable asynchronous start.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_EnableAsyncStart

LL_I2S_DisableAsyncStart

Function name

__STATIC_INLINE void LL_I2S_DisableAsyncStart (SPI_TypeDef * SPIx)

Function description

Disable asynchronous start.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_DisableAsyncStart

LL_I2S_IsEnabledAsyncStart

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledAsyncStart (SPI_TypeDef * SPIx)

Function description

Check if asynchronous start is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_IsEnabledAsyncStart

LL_I2S_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)

Function description

Check if Rx buffer is not empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_I2S_IsActiveFlag_RXNE

LL_I2S_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE (SPI_TypeDef * SPIx)

Function description

Check if Tx buffer is empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXE LL_I2S_IsActiveFlag_TXE

LL_I2S_IsActiveFlag_BSY

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY (SPI_TypeDef * SPIx)

Function description

Get busy flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSY LL_I2S_IsActiveFlag_BSY

LL_I2S_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (SPI_TypeDef * SPIx)

Function description

Get overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (SPI_TypeDef * SPIx)

Function description

Get underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (SPI_TypeDef * SPIx)

Function description

Get frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR FRE LL_I2S_IsActiveFlag_FRE

LL_I2S_IsActiveFlag_CHSIDE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_CHSIDE (SPI_TypeDef * SPIx)

Function description

Get channel side flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- 0: Channel Left has to be transmitted or has been received 1: Channel Right has to be transmitted or has been received It has no significance in PCM mode.

Reference Manual to LL API cross reference:

- SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE

LL_I2S_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR FRE LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)

Function description

Enable error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_EnableIT_ERR

LL_I2S_EnableIT_RXNE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Enable Rx buffer not empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_EnableIT_RXNE

LL_I2S_EnableIT_TXE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_TXE (SPI_TypeDef * SPIx)

Function description

Enable Tx buffer empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_EnableIT_TXE

LL_I2S_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_ERR (SPI_TypeDef * SPIx)

Function description

Disable error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_DisableIT_ERR

LL_I2S_DisableIT_RXNE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Disable Rx buffer not empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_DisableIT_RXNE

LL_I2S_DisableIT_TXE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_TXE (SPI_TypeDef * SPIx)

Function description

Disable Tx buffer empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_DisableIT_TXE

LL_I2S_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_ERR (SPI_TypeDef * SPIx)

Function description

Check if ERR IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_IsEnabledIT_ERR

LL_I2S_IsEnabledIT_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)

Function description

Check if RXNE IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_IsEnabledIT_RXNE

LL_I2S_IsEnabledIT_TXE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXE (SPI_TypeDef * SPIx)

Function description

Check if TXE IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_IsEnabledIT_TXE

LL_I2S_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read 16-Bits in data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- DR DR LL_I2S_ReceiveData16

LL_I2S_TransmitData16

Function name

__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write 16-Bits in data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_I2S_TransmitData16

LL_I2S_DeInit

Function name

ErrorStatus LL_I2S_DeInit (SPI_TypeDef * SPIx)

Function description

De-initialize the SPI/I2S registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_I2S_Init

Function name

ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_I2S_StructInit

Function name

void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Set each LL_I2S_InitTypeDef field to default value.

Parameters

- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_ConfigPrescaler

Function name

void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)

Function description

Set linear and parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** value Min_Data=0x02 and Max_Data=0xFF.
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Notes

- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).

88.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

88.3.1 SPI

SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_DIV2

BaudRate control equal to fPCLK/2

LL_SPI_BAUDRATEPRESCALER_DIV4

BaudRate control equal to fPCLK/4

LL_SPI_BAUDRATEPRESCALER_DIV8

BaudRate control equal to fPCLK/8

LL_SPI_BAUDRATEPRESCALER_DIV16

BaudRate control equal to fPCLK/16

LL_SPI_BAUDRATEPRESCALER_DIV32

BaudRate control equal to fPCLK/32

LL_SPI_BAUDRATEPRESCALER_DIV64

BaudRate control equal to fPCLK/64

LL_SPI_BAUDRATEPRESCALER_DIV128

BaudRate control equal to fPCLK/128

LL_SPI_BAUDRATEPRESCALER_DIV256

BaudRate control equal to fPCLK/256

Transmission Bit Order

LL_SPI_LSB_FIRST

Data is transmitted/received with the LSB first

LL_SPI_MSB_FIRST

Data is transmitted/received with the MSB first

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE

CRC calculation disabled

LL_SPI_CRCCALCULATION_ENABLE

CRC calculation enabled

CRC Length

LL_SPI_CRC_8BIT

8-bit CRC length

LL_SPI_CRC_16BIT

16-bit CRC length

Datawidth

LL_SPI_DATAWIDTH_4BIT

Data length for SPI transfer: 4 bits

LL_SPI_DATAWIDTH_5BIT

Data length for SPI transfer: 5 bits

LL_SPI_DATAWIDTH_6BIT

Data length for SPI transfer: 6 bits

LL_SPI_DATAWIDTH_7BIT

Data length for SPI transfer: 7 bits

LL_SPI_DATAWIDTH_8BIT

Data length for SPI transfer: 8 bits

LL_SPI_DATAWIDTH_9BIT

Data length for SPI transfer: 9 bits

LL_SPI_DATAWIDTH_10BIT

Data length for SPI transfer: 10 bits

LL_SPI_DATAWIDTH_11BIT

Data length for SPI transfer: 11 bits

LL_SPI_DATAWIDTH_12BIT

Data length for SPI transfer: 12 bits

LL_SPI_DATAWIDTH_13BIT

Data length for SPI transfer: 13 bits

LL_SPI_DATAWIDTH_14BIT

Data length for SPI transfer: 14 bits

LL_SPI_DATAWIDTH_15BIT

Data length for SPI transfer: 15 bits

LL_SPI_DATAWIDTH_16BIT

Data length for SPI transfer: 16 bits

DMA Parity

LL_SPI_DMA_PARITY_EVEN

Select DMA parity Even

LL_SPI_DMA_PARITY_ODD

Select DMA parity Odd

Get Flags Defines

LL_SPI_SR_RXNE

Rx buffer not empty flag

LL_SPI_SR_TXE

Tx buffer empty flag

LL_SPI_SR_BSY

Busy flag

LL_SPI_SR_CRCERR

CRC error flag

LL_SPI_SR_MODF

Mode fault flag

LL_SPI_SR_OVR

Overrun flag

LL_SPI_SR_FRE

TI mode frame format error flag

IT Defines

LL_SPI_CR2_RXNEIE

Rx buffer not empty interrupt enable

LL_SPI_CR2_TXEIE

Tx buffer empty interrupt enable

LL_SPI_CR2_ERRIE

Error interrupt enable

LL_I2S_CR2_RXNEIE

Rx buffer not empty interrupt enable

LL_I2S_CR2_TXEIE

Tx buffer empty interrupt enable

LL_I2S_CR2_ERRIE

Error interrupt enable

Operation Mode

LL_SPI_MODE_MASTER

Master configuration

LL_SPI_MODE_SLAVE

Slave configuration

Slave Select Pin Mode

LL_SPI_NSS_SOFT

NSS managed internally. NSS pin not used and free

LL_SPI_NSS_HARD_INPUT

NSS pin used in Input. Only used in Master mode

LL_SPI_NSS_HARD_OUTPUT

NSS pin used in Output. Only used in Slave mode as chip select

Clock Phase

LL_SPI_PHASE_1EDGE

First clock transition is the first data capture edge

LL_SPI_PHASE_2EDGE

Second clock transition is the first data capture edge

Clock Polarity

LL_SPI_POLARITY_LOW

Clock to 0 when idle

LL_SPI_POLARITY_HIGH

Clock to 1 when idle

Serial Protocol

LL_SPI_PROTOCOL_MOTOROLA

Motorola mode. Used as default value

LL_SPI_PROTOCOL_TI

TI mode

RX FIFO Level

LL_SPI_RX_FIFO_EMPTY

FIFO reception empty

LL_SPI_RX_FIFO_QUARTER_FULL

FIFO reception 1/4

LL_SPI_RX_FIFO_HALF_FULL

FIFO reception 1/2

LL_SPI_RX_FIFO_FULL

FIFO reception full

RX FIFO Threshold

LL_SPI_RX_FIFO_TH_HALF

RXNE event is generated if FIFO level is greater than or equel to 1/2 (16-bit)

LL_SPI_RX_FIFO_TH_QUARTER

RXNE event is generated if FIFO level is greater than or equel to 1/4 (8-bit)

Transfer Mode

LL_SPI_FULL_DUPLEX

Full-Duplex mode. Rx and Tx transfer on 2 lines

LL_SPI_SIMPLEX_RX

Simplex Rx mode. Rx transfer only on 1 line

LL_SPI_HALF_DUPLEX_RX

Half-Duplex Rx mode. Rx transfer on 1 line

LL_SPI_HALF_DUPLEX_TX

Half-Duplex Tx mode. Tx transfer on 1 line

TX FIFO Level

LL_SPI_TX_FIFO_EMPTY

FIFO transmission empty

LL_SPI_TX_FIFO_QUARTER_FULL

FIFO transmission 1/4

LL_SPI_TX_FIFO_HALF_FULL

FIFO transmission 1/2

LL_SPI_TX_FIFO_FULL

FIFO transmission full

Common Write and read registers Macros

LL_SPI_WriteReg

Description:

- Write a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SPI_ReadReg

Description:

- Read a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:

- Register: value

89 LL SYSTEM Generic Driver

89.1 SYSTEM Firmware driver API description

The following section lists the various functions of the SYSTEM library.

89.1.1 Detailed description of functions

LL_SYSCFG_SetRemapMemory

Function name

__STATIC_INLINE void LL_SYSCFG_SetRemapMemory (uint32_t Memory)

Function description

Set memory mapping at address 0x00000000.

Parameters

- **Memory:** This parameter can be one of the following values:
  - LL_SYSCFG_REMAP_FLASH
  - LL_SYSCFG_REMAP_SYSTEMFLASH
  - LL_SYSCFG_REMAP_SRAM
  - LL_SYSCFG_REMAP_FMC (*)
  - LL_SYSCFG_REMAP_QUADSPI (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_MEMRMP MEM_MODE LL_SYSCFG_SetRemapMemory

LL_SYSCFG_GetRemapMemory

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetRemapMemory (void)

Function description

Get memory mapping at address 0x00000000.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_REMAP_FLASH
  - LL_SYSCFG_REMAP_SYSTEMFLASH
  - LL_SYSCFG_REMAP_SRAM
  - LL_SYSCFG_REMAP_FMC (*)
  - LL_SYSCFG_REMAP_QUADSPI (*)

  (*) value not defined in all devices

Reference Manual to LL API cross reference:

- SYSCFG_MEMRMP MEM_MODE LL_SYSCFG_GetRemapMemory

LL_SYSCFG_SetFlashBankMode

Function name

__STATIC_INLINE void LL_SYSCFG_SetFlashBankMode (uint32_t Bank)

Function description

Select Flash bank mode (Bank flashed at 0x08000000)

Parameters

- **Bank:** This parameter can be one of the following values:
  - LL_SYSCFG_BANKMODE_BANK1
  - LL_SYSCFG_BANKMODE_BANK2

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_MEMRMP FB_MODE LL_SYSCFG_SetFlashBankMode

LL_SYSCFG_GetFlashBankMode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashBankMode (void)

Function description

Get Flash bank mode (Bank flashed at 0x08000000)

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_BANKMODE_BANK1
  - LL_SYSCFG_BANKMODE_BANK2

Reference Manual to LL API cross reference:

- SYSCFG_MEMRMP FB_MODE LL_SYSCFG_GetFlashBankMode

LL_SYSCFG_EnableAnalogBooster

Function name

__STATIC_INLINE void LL_SYSCFG_EnableAnalogBooster (void)

Function description

Enable I/O analog switch voltage booster.

Return values

- **None:**

Notes

- When voltage booster is enabled, I/O analog switches are supplied by a dedicated voltage booster, from VDD power domain. This is the recommended configuration with low VDDA voltage operation.
- The I/O analog switch voltage booster is relevant for peripherals using I/O in analog input: ADC, COMP, OPAMP. However, COMP and OPAMP inputs have a high impedance and voltage booster do not impact performance significantly. Therefore, the voltage booster is mainly intended for usage with ADC.

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 BOOSTEN LL_SYSCFG_EnableAnalogBooster

LL_SYSCFG_DisableAnalogBooster

Function name

__STATIC_INLINE void LL_SYSCFG_DisableAnalogBooster (void)

Function description

Disable I/O analog switch voltage booster.

Return values

- **None:**

Notes

- When voltage booster is enabled, I/O analog switches are supplied by a dedicated voltage booster, from VDD power domain. This is the recommended configuration with low VDDA voltage operation.
- The I/O analog switch voltage booster is relevant for peripherals using I/O in analog input: ADC, COMP, OPAMP. However, COMP and OPAMP inputs have a high impedance and voltage booster do not impact performance significantly. Therefore, the voltage booster is mainly intended for usage with ADC.

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 BOOSTEN LL_SYSCFG_DisableAnalogBooster

LL_SYSCFG_EnableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C3
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C4 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 I2C_PBx_FMP LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2Cx_FMP LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_DisableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C3
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C4 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 I2C_PBx_FMP LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2Cx_FMP LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_EnableIT_FPU_IOC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IOC (void)

Function description

Enable Floating Point Unit Invalid operation Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_0 LL_SYSCFG_EnableIT_FPU_IOC

LL_SYSCFG_EnableIT_FPU_DZC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_DZC (void)

Function description

Enable Floating Point Unit Divide-by-zero Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_1 LL_SYSCFG_EnableIT_FPU_DZC

LL_SYSCFG_EnableIT_FPU_UFC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_UFC (void)

Function description

Enable Floating Point Unit Underflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_2 LL_SYSCFG_EnableIT_FPU_UFC

LL_SYSCFG_EnableIT_FPU_OFC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_OFC (void)

Function description

Enable Floating Point Unit Overflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_3 LL_SYSCFG_EnableIT_FPU_OFC

LL_SYSCFG_EnableIT_FPU_IDC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IDC (void)

Function description

Enable Floating Point Unit Input denormal Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_4 LL_SYSCFG_EnableIT_FPU_IDC

LL_SYSCFG_EnableIT_FPU_IXC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IXC (void)

Function description

Enable Floating Point Unit Inexact Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_5 LL_SYSCFG_EnableIT_FPU_IXC

LL_SYSCFG_DisableIT_FPU_IOC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IOC (void)

Function description

Disable Floating Point Unit Invalid operation Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_0 LL_SYSCFG_DisableIT_FPU_IOC

LL_SYSCFG_DisableIT_FPU_DZC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_DZC (void)

Function description

Disable Floating Point Unit Divide-by-zero Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_1 LL_SYSCFG_DisableIT_FPU_DZC

LL_SYSCFG_DisableIT_FPU_UFC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_UFC (void)

Function description

Disable Floating Point Unit Underflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_2 LL_SYSCFG_DisableIT_FPU_UFC

LL_SYSCFG_DisableIT_FPU_OFC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_OFC (void)

Function description

Disable Floating Point Unit Overflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_3 LL_SYSCFG_DisableIT_FPU_OFC

LL_SYSCFG_DisableIT_FPU_IDC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IDC (void)

Function description

Disable Floating Point Unit Input denormal Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_4 LL_SYSCFG_DisableIT_FPU_IDC

LL_SYSCFG_DisableIT_FPU_IXC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IXC (void)

Function description

Disable Floating Point Unit Inexact Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_5 LL_SYSCFG_DisableIT_FPU_IXC

LL_SYSCFG_IsEnabledIT_FPU_IOC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IOC (void)

Function description

Check if Floating Point Unit Invalid operation Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_0 LL_SYSCFG_IsEnabledIT_FPU_IOC

LL_SYSCFG_IsEnabledIT_FPU_DZC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_DZC (void)

Function description

Check if Floating Point Unit Divide-by-zero Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_1 LL_SYSCFG_IsEnabledIT_FPU_DZC

LL_SYSCFG_IsEnabledIT_FPU_UFC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_UFC (void)

Function description

Check if Floating Point Unit Underflow Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_2 LL_SYSCFG_IsEnabledIT_FPU_UFC

LL_SYSCFG_IsEnabledIT_FPU_OFC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_OFC (void)

Function description

Check if Floating Point Unit Overflow Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_3 LL_SYSCFG_IsEnabledIT_FPU_OFC

LL_SYSCFG_IsEnabledIT_FPU_IDC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IDC (void)

Function description

Check if Floating Point Unit Input denormal Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_4 LL_SYSCFG_IsEnabledIT_FPU_IDC

LL_SYSCFG_IsEnabledIT_FPU_IXC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IXC (void)

Function description

Check if Floating Point Unit Inexact Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 FPU_IE_5 LL_SYSCFG_IsEnabledIT_FPU_IXC

LL_SYSCFG_SetEXTISource

Function name

__STATIC_INLINE void LL_SYSCFG_SetEXTISource (uint32_t Port, uint32_t Line)

Function description

Configure source input for the EXTI external interrupt.

Parameters

- **Port:** This parameter can be one of the following values:
  - LL_SYSCFG_EXTI_PORTA
  - LL_SYSCFG_EXTI_PORTB
  - LL_SYSCFG_EXTI_PORTC
  - LL_SYSCFG_EXTI_PORTD
  - LL_SYSCFG_EXTI_PORTE
  - LL_SYSCFG_EXTI_PORTF
  - LL_SYSCFG_EXTI_PORTG

  (*) value not defined in all devices
- **Line:** This parameter can be one of the following values:
  - LL_SYSCFG_EXTI_LINE0
  - LL_SYSCFG_EXTI_LINE1
  - LL_SYSCFG_EXTI_LINE2
  - LL_SYSCFG_EXTI_LINE3
  - LL_SYSCFG_EXTI_LINE4
  - LL_SYSCFG_EXTI_LINE5
  - LL_SYSCFG_EXTI_LINE6
  - LL_SYSCFG_EXTI_LINE7
  - LL_SYSCFG_EXTI_LINE8
  - LL_SYSCFG_EXTI_LINE9
  - LL_SYSCFG_EXTI_LINE10
  - LL_SYSCFG_EXTI_LINE11
  - LL_SYSCFG_EXTI_LINE12
  - LL_SYSCFG_EXTI_LINE13
  - LL_SYSCFG_EXTI_LINE14
  - LL_SYSCFG_EXTI_LINE15

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_EXTICR1 EXTIx LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR2 EXTIx LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR3 EXTIx LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR4 EXTIx LL_SYSCFG_SetEXTISource

LL_SYSCFG_GetEXTISource

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetEXTISource (uint32_t Line)

Function description

Get the configured defined for specific EXTI Line.

Parameters

- **Line:** This parameter can be one of the following values:
  - LL_SYSCFG_EXTI_LINE0
  - LL_SYSCFG_EXTI_LINE1
  - LL_SYSCFG_EXTI_LINE2
  - LL_SYSCFG_EXTI_LINE3
  - LL_SYSCFG_EXTI_LINE4
  - LL_SYSCFG_EXTI_LINE5
  - LL_SYSCFG_EXTI_LINE6
  - LL_SYSCFG_EXTI_LINE7
  - LL_SYSCFG_EXTI_LINE8
  - LL_SYSCFG_EXTI_LINE9
  - LL_SYSCFG_EXTI_LINE10
  - LL_SYSCFG_EXTI_LINE11
  - LL_SYSCFG_EXTI_LINE12
  - LL_SYSCFG_EXTI_LINE13
  - LL_SYSCFG_EXTI_LINE14
  - LL_SYSCFG_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_EXTI_PORTA
  - LL_SYSCFG_EXTI_PORTB
  - LL_SYSCFG_EXTI_PORTC
  - LL_SYSCFG_EXTI_PORTD
  - LL_SYSCFG_EXTI_PORTE
  - LL_SYSCFG_EXTI_PORTF
  - LL_SYSCFG_EXTI_PORTG

  (*) value not defined in all devices

Reference Manual to LL API cross reference:

- SYSCFG_EXTICR1 EXTIx LL_SYSCFG_GetEXTISource
- SYSCFG_EXTICR2 EXTIx LL_SYSCFG_GetEXTISource
- SYSCFG_EXTICR3 EXTIx LL_SYSCFG_GetEXTISource
- SYSCFG_EXTICR4 EXTIx LL_SYSCFG_GetEXTISource

LL_SYSCFG_EnableCCMSRAMErase

Function name

__STATIC_INLINE void LL_SYSCFG_EnableCCMSRAMErase (void)

Function description

Enable CCMSRAM Erase (starts a hardware CCMSRAM erase operation.

Return values

- **None:**

Notes

- This bit is write-protected: setting this bit is possible only after the correct key sequence is written in the SYSCFG_SKR register as described in the Reference Manual.

Reference Manual to LL API cross reference:

- SYSCFG_SCSR CCMER LL_SYSCFG_EnableCCMSRAMErase

LL_SYSCFG_IsCCMSRAMEraseOngoing

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsCCMSRAMEraseOngoing (void)

Function description

Check if CCMSRAM erase operation is on going.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_SCSR CCMBSY LL_SYSCFG_IsCCMSRAMEraseOngoing

LL_SYSCFG_SetTIMBreakInputs

Function name

__STATIC_INLINE void LL_SYSCFG_SetTIMBreakInputs (uint32_t Break)

Function description

Set connections to TIM1/8/15/16/17 Break inputs.

Parameters

- **Break:** This parameter can be a combination of the following values:
  - LL_SYSCFG_TIMBREAK_ECC
  - LL_SYSCFG_TIMBREAK_PVD
  - LL_SYSCFG_TIMBREAK_SRAM_PARITY
  - LL_SYSCFG_TIMBREAK_LOCKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 CLL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR2 SPL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR2 PVDL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR2 ECCL LL_SYSCFG_SetTIMBreakInputs

LL_SYSCFG_GetTIMBreakInputs

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetTIMBreakInputs (void)

Function description

Get connections to TIM1/8/15/16/17 Break inputs.

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_SYSCFG_TIMBREAK_ECC
  - LL_SYSCFG_TIMBREAK_PVD
  - LL_SYSCFG_TIMBREAK_SRAM_PARITY
  - LL_SYSCFG_TIMBREAK_LOCKUP

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 CLL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR2 SPL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR2 PVDL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR2 ECCL LL_SYSCFG_GetTIMBreakInputs

LL_SYSCFG_IsActiveFlag_SP

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SP (void)

Function description

Check if SRAM parity error detected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 SPF LL_SYSCFG_IsActiveFlag_SP

LL_SYSCFG_ClearFlag_SP

Function name

__STATIC_INLINE void LL_SYSCFG_ClearFlag_SP (void)

Function description

Clear SRAM parity error flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 SPF LL_SYSCFG_ClearFlag_SP

LL_SYSCFG_EnableCCMSRAMPageWRP

Function name

__STATIC_INLINE void LL_SYSCFG_EnableCCMSRAMPageWRP (uint32_t CCMSRAMWRP)

Function description

Enable CCMSRAM page write protection.

Parameters

- **CCMSRAMWRP:** This parameter can be a combination of the following values:
  - LL_SYSCFG_CCMSRAMWRP_PAGE0
  - LL_SYSCFG_CCMSRAMWRP_PAGE1
  - LL_SYSCFG_CCMSRAMWRP_PAGE2
  - LL_SYSCFG_CCMSRAMWRP_PAGE3
  - LL_SYSCFG_CCMSRAMWRP_PAGE4
  - LL_SYSCFG_CCMSRAMWRP_PAGE5
  - LL_SYSCFG_CCMSRAMWRP_PAGE6
  - LL_SYSCFG_CCMSRAMWRP_PAGE7
  - LL_SYSCFG_CCMSRAMWRP_PAGE8
  - LL_SYSCFG_CCMSRAMWRP_PAGE9
  - LL_SYSCFG_CCMSRAMWRP_PAGE10 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE11 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE12 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE13 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE14 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE15 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE16 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE17 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE18 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE19 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE20 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE21 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE22 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE23 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE24 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE25 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE26 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE27 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE28 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE29 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE30 (*)
  - LL_SYSCFG_CCMSRAMWRP_PAGE31 (*)

  (*) value not defined in all devices

Return values

- **None:**

Notes

- Write protection is cleared only by a system reset

Reference Manual to LL API cross reference:

- SYSCFG_SWPR PAGEx LL_SYSCFG_EnableCCMSRAMPageWRP

LL_SYSCFG_LockCCMSRAMWRP

Function name

__STATIC_INLINE void LL_SYSCFG_LockCCMSRAMWRP (void)

Function description

CCMSRAM page write protection lock prior to erase.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_SKR KEY LL_SYSCFG_LockCCMSRAMWRP

LL_SYSCFG_UnlockCCMSRAMWRP

Function name

__STATIC_INLINE void LL_SYSCFG_UnlockCCMSRAMWRP (void)

Function description

CCMSRAM page write protection unlock prior to erase.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_SKR KEY LL_SYSCFG_UnlockCCMSRAMWRP

LL_DBGMCU_GetDeviceID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)

Function description

Return the device identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0x0FFF (ex: device ID is 0x6415)

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)

Function description

Return the device revision identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- This field indicates the revision of the device.

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGSleepMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGSleepMode (void)

Function description

Enable the Debug Module during SLEEP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_SLEEP LL_DBGMCU_EnableDBGSleepMode

LL_DBGMCU_DisableDBGSleepMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGSleepMode (void)

Function description

Disable the Debug Module during SLEEP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_SLEEP LL_DBGMCU_DisableDBGSleepMode

LL_DBGMCU_EnableDBGStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode

LL_DBGMCU_DisableDBGStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode

LL_DBGMCU_EnableDBGStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

LL_DBGMCU_SetTracePinAssignment

Function name

__STATIC_INLINE void LL_DBGMCU_SetTracePinAssignment (uint32_t PinAssignment)

Function description

Set Trace pin assignment control.

Parameters

- **PinAssignment:** This parameter can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_SetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_SetTracePinAssignment

LL_DBGMCU_GetTracePinAssignment

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetTracePinAssignment (void)

Function description

Get Trace pin assignment control.

Return values

- **Returned:** value can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_GetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_GetTracePinAssignment

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP (*)
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_RTC_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_I2C3_STOP
  - LL_DBGMCU_APB1_GRP1_LPTIM1_STOP

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph

LL_DBGMCU_APB1_GRP2_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group2 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_I2C4_STOP (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP (*)
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_RTC_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_I2C3_STOP
  - LL_DBGMCU_APB1_GRP1_LPTIM1_STOP

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group2 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_I2C4_STOP (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP
  - LL_DBGMCU_APB2_GRP1_TIM20_STOP (*)
  - LL_DBGMCU_APB2_GRP1_HRTIM1_STOP (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZ DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP
  - LL_DBGMCU_APB2_GRP1_TIM20_STOP (*)
  - LL_DBGMCU_APB2_GRP1_HRTIM1_STOP (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZ DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_UnFreezePeriph

LL_VREFBUF_Enable

Function name

__STATIC_INLINE void LL_VREFBUF_Enable (void)

Function description

Enable Internal voltage reference.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR ENVR LL_VREFBUF_Enable

LL_VREFBUF_Disable

Function name

__STATIC_INLINE void LL_VREFBUF_Disable (void)

Function description

Disable Internal voltage reference.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR ENVR LL_VREFBUF_Disable

LL_VREFBUF_EnableHIZ

Function name

__STATIC_INLINE void LL_VREFBUF_EnableHIZ (void)

Function description

Enable high impedance (VREF+pin is high impedance)

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR HIZ LL_VREFBUF_EnableHIZ

LL_VREFBUF_DisableHIZ

Function name

__STATIC_INLINE void LL_VREFBUF_DisableHIZ (void)

Function description

Disable high impedance (VREF+pin is internally connected to the voltage reference buffer output)

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR HIZ LL_VREFBUF_DisableHIZ

LL_VREFBUF_SetVoltageScaling

Function name

__STATIC_INLINE void LL_VREFBUF_SetVoltageScaling (uint32_t Scale)

Function description

Set the Voltage reference scale.

Parameters

- **Scale:** This parameter can be one of the following values:
  - LL_VREFBUF_VOLTAGE_SCALE0
  - LL_VREFBUF_VOLTAGE_SCALE1
  - LL_VREFBUF_VOLTAGE_SCALE2

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRS LL_VREFBUF_SetVoltageScaling

LL_VREFBUF_GetVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_GetVoltageScaling (void)

Function description

Get the Voltage reference scale.

Return values

- **Returned:** value can be one of the following values:
  - LL_VREFBUF_VOLTAGE_SCALE0
  - LL_VREFBUF_VOLTAGE_SCALE1
  - LL_VREFBUF_VOLTAGE_SCALE2

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRS LL_VREFBUF_GetVoltageScaling

LL_VREFBUF_IsVREFReady

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_IsVREFReady (void)

Function description

Check if Voltage reference buffer is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRR LL_VREFBUF_IsVREFReady

LL_VREFBUF_GetTrimming

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_GetTrimming (void)

Function description

Get the trimming code for VREFBUF calibration.

Return values

- **Between:** 0 and 0x3F

Reference Manual to LL API cross reference:

- VREFBUF_CCR TRIM LL_VREFBUF_GetTrimming

LL_VREFBUF_SetTrimming

Function name

__STATIC_INLINE void LL_VREFBUF_SetTrimming (uint32_t Value)

Function description

Set the trimming code for VREFBUF calibration (Tune the internal reference buffer voltage)

Parameters

- **Value:** Between 0 and 0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CCR TRIM LL_VREFBUF_SetTrimming

LL_FLASH_SetLatency

Function name

__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)

Function description

Set FLASH Latency.

Parameters

- **Latency:** This parameter can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5 (*)
  - LL_FLASH_LATENCY_6 (*)
  - LL_FLASH_LATENCY_7 (*)
  - LL_FLASH_LATENCY_8 (*)
  - LL_FLASH_LATENCY_9 (*)
  - LL_FLASH_LATENCY_10 (*)
  - LL_FLASH_LATENCY_11 (*)
  - LL_FLASH_LATENCY_12 (*)
  - LL_FLASH_LATENCY_13 (*)
  - LL_FLASH_LATENCY_14 (*)
  - LL_FLASH_LATENCY_15 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)

Function description

Get FLASH Latency.

Return values

- **Returned:** value can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5 (*)
  - LL_FLASH_LATENCY_6 (*)
  - LL_FLASH_LATENCY_7 (*)
  - LL_FLASH_LATENCY_8 (*)
  - LL_FLASH_LATENCY_9 (*)
  - LL_FLASH_LATENCY_10 (*)
  - LL_FLASH_LATENCY_11 (*)
  - LL_FLASH_LATENCY_12 (*)
  - LL_FLASH_LATENCY_13 (*)
  - LL_FLASH_LATENCY_14 (*)
  - LL_FLASH_LATENCY_15 (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_FLASH_EnablePrefetch

Function name

__STATIC_INLINE void LL_FLASH_EnablePrefetch (void)

Function description

Enable Prefetch.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR PRFTEN LL_FLASH_EnablePrefetch

LL_FLASH_DisablePrefetch

Function name

__STATIC_INLINE void LL_FLASH_DisablePrefetch (void)

Function description

Disable Prefetch.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR PRFTEN LL_FLASH_DisablePrefetch

LL_FLASH_IsPrefetchEnabled

Function name

__STATIC_INLINE uint32_t LL_FLASH_IsPrefetchEnabled (void)

Function description

Check if Prefetch buffer is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FLASH_ACR PRFTEN LL_FLASH_IsPrefetchEnabled

LL_FLASH_EnableInstCache

Function name

__STATIC_INLINE void LL_FLASH_EnableInstCache (void)

Function description

Enable Instruction cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR ICEN LL_FLASH_EnableInstCache

LL_FLASH_DisableInstCache

Function name

__STATIC_INLINE void LL_FLASH_DisableInstCache (void)

Function description

Disable Instruction cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR ICEN LL_FLASH_DisableInstCache

LL_FLASH_EnableDataCache

Function name

__STATIC_INLINE void LL_FLASH_EnableDataCache (void)

Function description

Enable Data cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR DCEN LL_FLASH_EnableDataCache

LL_FLASH_DisableDataCache

Function name

__STATIC_INLINE void LL_FLASH_DisableDataCache (void)

Function description

Disable Data cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR DCEN LL_FLASH_DisableDataCache

LL_FLASH_EnableInstCacheReset

Function name

__STATIC_INLINE void LL_FLASH_EnableInstCacheReset (void)

Function description

Enable Instruction cache reset.

Return values

- **None:**

Notes

- bit can be written only when the instruction cache is disabled

Reference Manual to LL API cross reference:

- FLASH_ACR ICRST LL_FLASH_EnableInstCacheReset

LL_FLASH_DisableInstCacheReset

Function name

__STATIC_INLINE void LL_FLASH_DisableInstCacheReset (void)

Function description

Disable Instruction cache reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR ICRST LL_FLASH_DisableInstCacheReset

LL_FLASH_EnableDataCacheReset

Function name

__STATIC_INLINE void LL_FLASH_EnableDataCacheReset (void)

Function description

Enable Data cache reset.

Return values

- **None:**

Notes

- bit can be written only when the data cache is disabled

Reference Manual to LL API cross reference:

- FLASH_ACR DCRST LL_FLASH_EnableDataCacheReset

LL_FLASH_DisableDataCacheReset

Function name

__STATIC_INLINE void LL_FLASH_DisableDataCacheReset (void)

Function description

Disable Data cache reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR DCRST LL_FLASH_DisableDataCacheReset

LL_FLASH_EnableRunPowerDown

Function name

__STATIC_INLINE void LL_FLASH_EnableRunPowerDown (void)

Function description

Enable Flash Power-down mode during run mode or Low-power run mode.

Return values

- **None:**

Notes

- Flash memory can be put in power-down mode only when the code is executed from RAM
- Flash must not be accessed when power down is enabled
- Flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to LL API cross reference:

- FLASH_ACR RUN_PD LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY1 LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY2 LL_FLASH_EnableRunPowerDown

LL_FLASH_DisableRunPowerDown

Function name

__STATIC_INLINE void LL_FLASH_DisableRunPowerDown (void)

Function description

Disable Flash Power-down mode during run mode or Low-power run mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR RUN_PD LL_FLASH_DisableRunPowerDown
- FLASH_PDKEYR PDKEY1 LL_FLASH_DisableRunPowerDown
- FLASH_PDKEYR PDKEY2 LL_FLASH_DisableRunPowerDown

LL_FLASH_EnableSleepPowerDown

Function name

__STATIC_INLINE void LL_FLASH_EnableSleepPowerDown (void)

Function description

Enable Flash Power-down mode during Sleep or Low-power sleep mode.

Return values

- **None:**

Notes

- Flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to LL API cross reference:

- FLASH_ACR SLEEP_PD LL_FLASH_EnableSleepPowerDown

LL_FLASH_DisableSleepPowerDown

Function name

__STATIC_INLINE void LL_FLASH_DisableSleepPowerDown (void)

Function description

Disable Flash Power-down mode during Sleep or Low-power sleep mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR SLEEP_PD LL_FLASH_DisableSleepPowerDown

89.2 SYSTEM Firmware driver defines

The following section lists the various define and macros of the module.

89.2.1 SYSTEM

SYSTEM

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP

The counter clock of TIM2 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM3_STOP

The counter clock of TIM3 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM4_STOP

The counter clock of TIM4 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM5_STOP

The counter clock of TIM5 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM6_STOP

The counter clock of TIM6 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM7_STOP

The counter clock of TIM7 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_RTC_STOP

The clock of the RTC counter is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_WWDG_STOP

The window watchdog counter clock is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_IWDG_STOP

The independent watchdog counter clock is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_I2C1_STOP

The I2C1 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_I2C2_STOP

The I2C2 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_I2C3_STOP

The I2C3 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_LPTIM1_STOP

The counter clock of LPTIM1 is stopped when the core is halted

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_I2C4_STOP

The I2C4 SMBus timeout is frozen

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP

The counter clock of TIM1 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM8_STOP

The counter clock of TIM8 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM15_STOP

The counter clock of TIM15 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM16_STOP

The counter clock of TIM16 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM17_STOP

The counter clock of TIM17 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM20_STOP

The counter clock of TIM20 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_HRTIM1_STOP

The counter clock of HRTIM1 is stopped when the core is halted

SYSCFG BANK MODE

LL_SYSCFG_BANKMODE_BANK1

Flash Bank1 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank2 mapped at 0x08040000 (and aliased at 0x00080000)

LL_SYSCFG_BANKMODE_BANK2

Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08040000 (and aliased at 0x00080000)

SYSCFG CCMSRAM WRP

LL_SYSCFG_CCMSRAMWRP_PAGE0

CCMSRAM Write protection page 0

LL_SYSCFG_CCMSRAMWRP_PAGE1

CCMSRAM Write protection page 1

LL_SYSCFG_CCMSRAMWRP_PAGE2

CCMSRAM Write protection page 2

LL_SYSCFG_CCMSRAMWRP_PAGE3

CCMSRAM Write protection page 3

LL_SYSCFG_CCMSRAMWRP_PAGE4

CCMSRAM Write protection page 4

LL_SYSCFG_CCMSRAMWRP_PAGE5

CCMSRAM Write protection page 5

LL_SYSCFG_CCMSRAMWRP_PAGE6

CCMSRAM Write protection page 6

LL_SYSCFG_CCMSRAMWRP_PAGE7

CCMSRAM Write protection page 7

LL_SYSCFG_CCMSRAMWRP_PAGE8

CCMSRAM Write protection page 8

LL_SYSCFG_CCMSRAMWRP_PAGE9

CCMSRAM Write protection page 9

LL_SYSCFG_CCMSRAMWRP_PAGE10

CCMSRAM Write protection page 10

LL_SYSCFG_CCMSRAMWRP_PAGE11

CCMSRAM Write protection page 11

LL_SYSCFG_CCMSRAMWRP_PAGE12

CCMSRAM Write protection page 12

LL_SYSCFG_CCMSRAMWRP_PAGE13

CCMSRAM Write protection page 13

LL_SYSCFG_CCMSRAMWRP_PAGE14

CCMSRAM Write protection page 14

LL_SYSCFG_CCMSRAMWRP_PAGE15

CCMSRAM Write protection page 15

LL_SYSCFG_CCMSRAMWRP_PAGE16

CCMSRAM Write protection page 16

LL_SYSCFG_CCMSRAMWRP_PAGE17

CCMSRAM Write protection page 17

LL_SYSCFG_CCMSRAMWRP_PAGE18

CCMSRAM Write protection page 18

LL_SYSCFG_CCMSRAMWRP_PAGE19

CCMSRAM Write protection page 19

LL_SYSCFG_CCMSRAMWRP_PAGE20

CCMSRAM Write protection page 20

LL_SYSCFG_CCMSRAMWRP_PAGE21

CCMSRAM Write protection page 21

LL_SYSCFG_CCMSRAMWRP_PAGE22

CCMSRAM Write protection page 22

LL_SYSCFG_CCMSRAMWRP_PAGE23

CCMSRAM Write protection page 23

LL_SYSCFG_CCMSRAMWRP_PAGE24

CCMSRAM Write protection page 24

LL_SYSCFG_CCMSRAMWRP_PAGE25

CCMSRAM Write protection page 25

LL_SYSCFG_CCMSRAMWRP_PAGE26

CCMSRAM Write protection page 26

LL_SYSCFG_CCMSRAMWRP_PAGE27

CCMSRAM Write protection page 27

LL_SYSCFG_CCMSRAMWRP_PAGE28

CCMSRAM Write protection page 28

LL_SYSCFG_CCMSRAMWRP_PAGE29

CCMSRAM Write protection page 29

LL_SYSCFG_CCMSRAMWRP_PAGE30

CCMSRAM Write protection page 30

LL_SYSCFG_CCMSRAMWRP_PAGE31

CCMSRAM Write protection page 31

SYSCFG EXTI LINE

LL_SYSCFG_EXTI_LINE0

LL_SYSCFG_EXTI_LINE1

LL_SYSCFG_EXTI_LINE2

LL_SYSCFG_EXTI_LINE3

LL_SYSCFG_EXTI_LINE4

LL_SYSCFG_EXTI_LINE5

LL_SYSCFG_EXTI_LINE6

LL_SYSCFG_EXTI_LINE7

LL_SYSCFG_EXTI_LINE8

LL_SYSCFG_EXTI_LINE9

LL_SYSCFG_EXTI_LINE10

LL_SYSCFG_EXTI_LINE11

LL_SYSCFG_EXTI_LINE12

LL_SYSCFG_EXTI_LINE13

LL_SYSCFG_EXTI_LINE14

LL_SYSCFG_EXTI_LINE15

SYSCFG EXTI PORT

LL_SYSCFG_EXTI_PORTA
EXTI PORT A

LL_SYSCFG_EXTI_PORTB
EXTI PORT B

LL_SYSCFG_EXTI_PORTC
EXTI PORT C

LL_SYSCFG_EXTI_PORTD
EXTI PORT D

LL_SYSCFG_EXTI_PORTE
EXTI PORT E

LL_SYSCFG_EXTI_PORTF
EXTI PORT F

LL_SYSCFG_EXTI_PORTG
EXTI PORT G

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_PB6
Enable Fast Mode Plus on PB6

LL_SYSCFG_I2C_FASTMODEPLUS_PB7
Enable Fast Mode Plus on PB7

LL_SYSCFG_I2C_FASTMODEPLUS_PB8
Enable Fast Mode Plus on PB8

LL_SYSCFG_I2C_FASTMODEPLUS_PB9
Enable Fast Mode Plus on PB9

LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
Enable Fast Mode Plus on I2C1 pins

LL_SYSCFG_I2C_FASTMODEPLUS_I2C2
Enable Fast Mode Plus on I2C2 pins

LL_SYSCFG_I2C_FASTMODEPLUS_I2C3

Enable Fast Mode Plus on I2C3 pins

LL_SYSCFG_I2C_FASTMODEPLUS_I2C4

Enable Fast Mode Plus on I2C4 pins

FLASH LATENCY

LL_FLASH_LATENCY_0

FLASH Zero wait state

LL_FLASH_LATENCY_1

FLASH One wait state

LL_FLASH_LATENCY_2

FLASH Two wait states

LL_FLASH_LATENCY_3

FLASH Three wait states

LL_FLASH_LATENCY_4

FLASH Four wait states

LL_FLASH_LATENCY_5

FLASH five wait state

LL_FLASH_LATENCY_6

FLASH six wait state

LL_FLASH_LATENCY_7

FLASH seven wait states

LL_FLASH_LATENCY_8

FLASH eight wait states

LL_FLASH_LATENCY_9

FLASH nine wait states

LL_FLASH_LATENCY_10

FLASH ten wait states

LL_FLASH_LATENCY_11

FLASH eleven wait states

LL_FLASH_LATENCY_12

FLASH twelve wait states

LL_FLASH_LATENCY_13

FLASH thirteen wait states

LL_FLASH_LATENCY_14

FLASH fourteen wait states

LL_FLASH_LATENCY_15

FLASH fifteen wait states

SYSCFG REMAP

LL_SYSCFG_REMAP_FLASH

Main Flash memory mapped at 0x00000000

LL_SYSCFG_REMAP_SYSTEMFLASH

System Flash memory mapped at 0x00000000

LL_SYSCFG_REMAP_SRAM

SRAM1 mapped at 0x00000000

LL_SYSCFG_REMAP_FMC

FMC bank 1 (NOR/PSRAM 1 and 2) mapped at 0x00000000

LL_SYSCFG_REMAP_QUADSPI

QUADSPI memory mapped at 0x00000000

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_ECC

Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17

LL_SYSCFG_TIMBREAK_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface

LL_SYSCFG_TIMBREAK_SRAM_PARITY

Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/8/15/16/17

LL_SYSCFG_TIMBREAK_LOCKUP

Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/15/16/17

DBGMCU TRACE Pin Assignment

LL_DBGMCU_TRACE_NONE

TRACE pins not assigned (default state)

LL_DBGMCU_TRACE_ASYNCH

TRACE pin assignment for Asynchronous Mode

LL_DBGMCU_TRACE_SYNCH_SIZE1

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1

LL_DBGMCU_TRACE_SYNCH_SIZE2

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2

LL_DBGMCU_TRACE_SYNCH_SIZE4

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

VREFBUF VOLTAGE

LL_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREFBUF_OUT = 2.048V)

LL_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREFBUF_OUT = 2.5V)

LL_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 2 (VREFBUF_OUT = 2.9V)

90 LL TIM Generic Driver

90.1 TIM Firmware driver registers structures

90.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32g4xx_ll_tim.h

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_InitTypeDef::Prescaler***

  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***

  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***

  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***

  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint32_t LL_TIM_InitTypeDef::RepetitionCounter***

  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

90.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32g4xx_ll_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***

  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.

- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***

  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.

- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***

  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.

- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***

  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).

- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***

  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.

- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***

  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.

- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***

  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***

  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

90.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32g4xx_ll_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***

  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.

- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***

  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***

  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***

  Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

90.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32g4xx_ll_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***

  Specifies the encoder resolution (x2 or x4). This parameter can be a value of ***TIM_LL_EC_ENCODERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***

  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***

  Specifies the TI1 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***

  Specifies the TI1 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***

  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***

  Specifies the active edge of TI2 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***

  Specifies the TI2 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***

  Specifies the TI2 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***

  Specifies the TI2 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

90.1.5 LL_TIM_HALLSENSOR_InitTypeDef

LL_TIM_HALLSENSOR_InitTypeDef is defined in the stm32g4xx_ll_tim.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***

  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***

  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***

  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***

  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

90.1.6 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32g4xx_ll_tim.h

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***

  Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_LL_EC_OSSR***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**

  Note:

  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***

  Specifies the Off-State used in Idle state. This parameter can be a value of ***TIM_LL_EC_OSSI***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**

  Note:

  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***

  Specifies the LOCK level parameters. This parameter can be a value of ***TIM_LL_EC_LOCKLEVEL***

  Note:

  – The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.

- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***

  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.

- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***

  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***

  Specifies the TIM Break Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakFilter***

  Specifies the TIM Break Filter. This parameter can be a value of ***TIM_LL_EC_BREAK_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakAFMode***

  Specifies the alternate function mode of the break input. This parameter can be a value of ***TIM_LL_EC_BREAK_AFMODE***This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK()**

  Note:

  – Bidirectional break input is only supported by advanced timers instances.

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2State***

  Specifies whether the TIM Break2 input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK2_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK2()** or **LL_TIM_DisableBRK2()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Polarity***

  Specifies the TIM Break2 Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK2_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Filter***

  Specifies the TIM Break2 Filter. This parameter can be a value of ***TIM_LL_EC_BREAK2_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input. This parameter can be a value of ***TIM_LL_EC_BREAK2_AFMODE***This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK2()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***
  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of ***TIM_LL_EC_AUTOMATICOUTPUT_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

90.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

90.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name

__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)

Function description

Enable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name

__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)

Function description

Disable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (TIM_TypeDef * TIMx)

Function description

Indicates whether the timer counter is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Enable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Disable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (TIM_TypeDef * TIMx)

Function description

Indicates whether update event generation is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **Inverted:** state of bit (0 or 1).

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name

__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)

Function description

Set update event source.

Parameters

- **TIMx:** Timer instance
- **UpdateSource:** This parameter can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Return values

- **None:**

Notes

- Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller
- Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (TIM_TypeDef * TIMx)

Function description

Get actual event update source.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name

__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)

Function description

Set one pulse mode (one shot v.s.

Parameters

- **TIMx:** Timer instance
- **OnePulseMode:** This parameter can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (TIM_TypeDef * TIMx)

Function description

Get actual one pulse mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)

Function description

Set the timer counter counting mode.

Parameters

- **TIMx:** Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_SetCounterMode
- CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (TIM_TypeDef * TIMx)

Function description

Get actual counter mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetCounterMode
- CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name

__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)

Function description

Enable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name

__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)

Function description

Disable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (TIM_TypeDef * TIMx)

Function description

Indicates whether auto-reload (ARR) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name

__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)

Function description

Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance
- **ClockDivision:** This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name

__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (TIM_TypeDef * TIMx)

Function description

Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name

__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)

Function description

Set the counter value.

Parameters

- **TIMx:** Timer instance
- **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Return values

- **None:**

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounter (TIM_TypeDef * TIMx)

Function description

Get the counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetDirection (TIM_TypeDef * TIMx)

Function description

Get the current direction of the counter.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERDIRECTION_UP
  - LL_TIM_COUNTERDIRECTION_DOWN

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)

Function description

Set the prescaler value.

Parameters

- **TIMx:** Timer instance
- **Prescaler:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
- The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.
- Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (TIM_TypeDef * TIMx)

Function description

Get the prescaler value.

Parameters

- **TIMx:** Timer instance

Return values

- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name

__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)

Function description

Set the auto-reload value.

Parameters

- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter is blocked while the auto-reload value is null.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter In case dithering is activated,macro __LL_TIM_CALC_ARR_DITHER can be used instead, to calculate the AutoReload parameter.

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (TIM_TypeDef * TIMx)

Function description

Get the auto-reload value.

Parameters

- **TIMx:** Timer instance

Return values

- **Auto-reload:** value

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the returned value interpretation

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name

__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)

Function description

Set the repetition counter value.

Parameters

- **TIMx:** Timer instance
- **RepetitionCounter:** between Min_Data=0 and Max_Data=255 or 65535 for advanced timer.

Return values

- **None:**

Notes

- For advanced timer instances RepetitionCounter can be up to 65535.
- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (TIM_TypeDef * TIMx)

Function description

Get the repetition counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Repetition:** counter value

Notes

- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_EnableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_EnableUIFRemap (TIM_TypeDef * TIMx)

Function description

Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_EnableUIFRemap

LL_TIM_DisableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_DisableUIFRemap (TIM_TypeDef * TIMx)

Function description

Disable update interrupt flag (UIF) remapping.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_DisableUIFRemap

LL_TIM_IsActiveUIFCPY

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveUIFCPY (uint32_t Counter)

Function description

Indicate whether update interrupt flag (UIF) copy is set.

Parameters

- **Counter:** Counter value

Return values

- **State:** of bit (1 or 0).

LL_TIM_EnableDithering

Function name

__STATIC_INLINE void LL_TIM_EnableDithering (TIM_TypeDef * TIMx)

Function description

Enable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_EnableDithering

LL_TIM_DisableDithering

Function name

__STATIC_INLINE void LL_TIM_DisableDithering (TIM_TypeDef * TIMx)

Function description

Disable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_DisableDithering

LL_TIM_IsEnabledDithering

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDithering (TIM_TypeDef * TIMx)

Function description

Indicates whether dithering is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_IsEnabledDithering

LL_TIM_CC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)

Function description

Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs.
- Only on channels that have a complementary output.
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)

Function description

Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_SetUpdate

Function name

__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)

Function description

Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).

Parameters

- **TIMx:** Timer instance
- **CCUpdateSource:** This parameter can be one of the following values:
  - LL_TIM_CCUPDATESOURCE_COMG_ONLY
  - LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name

__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)

Function description

Set the trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance
- **DMAReqTrigger:** This parameter can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (TIM_TypeDef * TIMx)

Function description

Get actual trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name

__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)

Function description

Set the lock level to freeze the configuration of several capture/compare parameters.

Parameters

- **TIMx:** Timer instance
- **LockLevel:** This parameter can be one of the following values:
  - LL_TIM_LOCKLEVEL_OFF
  - LL_TIM_LOCKLEVEL_1
  - LL_TIM_LOCKLEVEL_2
  - LL_TIM_LOCKLEVEL_3

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.

Reference Manual to LL API cross reference:

- BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Enable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_EnableChannel
- CCER CC1NE LL_TIM_CC_EnableChannel
- CCER CC2E LL_TIM_CC_EnableChannel
- CCER CC2NE LL_TIM_CC_EnableChannel
- CCER CC3E LL_TIM_CC_EnableChannel
- CCER CC3NE LL_TIM_CC_EnableChannel
- CCER CC4E LL_TIM_CC_EnableChannel
- CCER CC4NE LL_TIM_CC_EnableChannel
- CCER CC5E LL_TIM_CC_EnableChannel
- CCER CC6E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Disable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel
- CCER CC4NE LL_TIM_CC_DisableChannel
- CCER CC5E LL_TIM_CC_DisableChannel
- CCER CC6E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Indicate whether channel(s) is(are) enabled.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel
- CCER CC4NE LL_TIM_CC_IsEnabledChannel
- CCER CC5E LL_TIM_CC_IsEnabledChannel
- CCER CC6E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name

__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
  - LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_OC_ConfigOutput
- CCMR1 CC2S LL_TIM_OC_ConfigOutput
- CCMR2 CC3S LL_TIM_OC_ConfigOutput
- CCMR2 CC4S LL_TIM_OC_ConfigOutput
- CCMR3 CC5S LL_TIM_OC_ConfigOutput
- CCMR3 CC6S LL_TIM_OC_ConfigOutput
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P LL_TIM_OC_ConfigOutput
- CCER CC5P LL_TIM_OC_ConfigOutput
- CCER CC6P LL_TIM_OC_ConfigOutput
- CR2 OIS1 LL_TIM_OC_ConfigOutput
- CR2 OIS2 LL_TIM_OC_ConfigOutput
- CR2 OIS3 LL_TIM_OC_ConfigOutput
- CR2 OIS4 LL_TIM_OC_ConfigOutput
- CR2 OIS5 LL_TIM_OC_ConfigOutput
- CR2 OIS6 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name

__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)

Function description

Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Mode:** This parameter can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - LL_TIM_OCMODE_ASSYMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode
- CCMR3 OC5M LL_TIM_OC_SetMode
- CCMR3 OC6M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the output compare mode of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - LL_TIM_OCMODE_ASSYMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_GetMode
- CCMR1 OC2M LL_TIM_OC_GetMode
- CCMR2 OC3M LL_TIM_OC_GetMode
- CCMR2 OC4M LL_TIM_OC_GetMode
- CCMR3 OC5M LL_TIM_OC_GetMode
- CCMR3 OC6M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity
- CCER CC4NP LL_TIM_OC_SetPolarity
- CCER CC5P LL_TIM_OC_SetPolarity
- CCER CC6P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity
- CCER CC4NP LL_TIM_OC_GetPolarity
- CCER CC5P LL_TIM_OC_GetPolarity
- CCER CC6P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name

__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)

Function description

Set the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **IdleState:** This parameter can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS2 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState
- CR2 OIS4N LL_TIM_OC_SetIdleState
- CR2 OIS5 LL_TIM_OC_SetIdleState
- CR2 OIS6 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS2 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS3 LL_TIM_OC_GetIdleState
- CR2 OIS3N LL_TIM_OC_GetIdleState
- CR2 OIS4 LL_TIM_OC_GetIdleState
- CR2 OIS4N LL_TIM_OC_GetIdleState
- CR2 OIS5 LL_TIM_OC_GetIdleState
- CR2 OIS6 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name

__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Acts only if the channel is configured in PWM1 or PWM2 mode.

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_EnableFast
- CCMR1 OC2FE LL_TIM_OC_EnableFast
- CCMR2 OC3FE LL_TIM_OC_EnableFast
- CCMR2 OC4FE LL_TIM_OC_EnableFast
- CCMR3 OC5FE LL_TIM_OC_EnableFast
- CCMR3 OC6FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name

__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_DisableFast
- CCMR1 OC2FE LL_TIM_OC_DisableFast
- CCMR2 OC3FE LL_TIM_OC_DisableFast
- CCMR2 OC4FE LL_TIM_OC_DisableFast
- CCMR3 OC5FE LL_TIM_OC_DisableFast
- CCMR3 OC6FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether fast mode is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_IsEnabledFast
- CCMR1 OC2FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC3FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC4FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC5FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC6FE LL_TIM_OC_IsEnabledFast

LL_TIM_OC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_EnablePreload
- CCMR1 OC2PE LL_TIM_OC_EnablePreload
- CCMR2 OC3PE LL_TIM_OC_EnablePreload
- CCMR2 OC4PE LL_TIM_OC_EnablePreload
- CCMR3 OC5PE LL_TIM_OC_EnablePreload
- CCMR3 OC6PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_DisablePreload
- CCMR1 OC2PE LL_TIM_OC_DisablePreload
- CCMR2 OC3PE LL_TIM_OC_DisablePreload
- CCMR2 OC4PE LL_TIM_OC_DisablePreload
- CCMR3 OC5PE LL_TIM_OC_DisablePreload
- CCMR3 OC6PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload
- CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC5PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC6PE LL_TIM_OC_IsEnabledPreload

LL_TIM_OC_EnableClear

Function name

__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_EnableClear
- CCMR1 OC2CE LL_TIM_OC_EnableClear
- CCMR2 OC3CE LL_TIM_OC_EnableClear
- CCMR2 OC4CE LL_TIM_OC_EnableClear
- CCMR3 OC5CE LL_TIM_OC_EnableClear
- CCMR3 OC6CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name

__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_DisableClear
- CCMR1 OC2CE LL_TIM_OC_DisableClear
- CCMR2 OC3CE LL_TIM_OC_DisableClear
- CCMR2 OC4CE LL_TIM_OC_DisableClear
- CCMR3 OC5CE LL_TIM_OC_DisableClear
- CCMR3 OC6CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates clearing the output channel on an external event is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Notes

- This function enables clearing the output channel on an external event.
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_IsEnabledClear
- CCMR1 OC2CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC3CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC4CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC5CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC6CE LL_TIM_OC_IsEnabledClear

LL_TIM_OC_SetDeadTime

Function name

__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge of the Ocx and OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter

Reference Manual to LL API cross reference:

- BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 1 (TIMx_CCR1).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 2 (TIMx_CCR2).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 3 (TIMx_CCR3).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 4 (TIMx_CCR4).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_SetCompareCH5

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH5 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 5 (TIMx_CCR5).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_SetCompareCH5

LL_TIM_OC_SetCompareCH6

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH6 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 6 (TIMx_CCR6).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_SetCompareCH6

LL_TIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR1) set for output channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR2) set for output channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR3) set for output channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR4) set for output channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_OC_GetCompareCH5

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH5 (TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR5) set for output channel 5.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_GetCompareCH5

LL_TIM_OC_GetCompareCH6

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH6 (TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR6) set for output channel 6.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_GetCompareCH6

LL_TIM_SetCH5CombinedChannels

Function name

__STATIC_INLINE void LL_TIM_SetCH5CombinedChannels (TIM_TypeDef * TIMx, uint32_t GroupCH5)

Function description

Select on which reference signal the OC5REF is combined to.

Parameters

- **TIMx:** Timer instance
- **GroupCH5:** This parameter can be a combination of the following values:
  - LL_TIM_GROUPCH5_NONE
  - LL_TIM_GROUPCH5_OC1REFC
  - LL_TIM_GROUPCH5_OC2REFC
  - LL_TIM_GROUPCH5_OC3REFC

Return values

- **None:**

Notes

- Macro IS_TIM_COMBINED3PHASEPWM_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the combined 3-phase PWM mode.

Reference Manual to LL API cross reference:

- CCR5 GC5C3 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C2 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C1 LL_TIM_SetCH5CombinedChannels

LL_TIM_OC_SetPulseWidthPrescaler

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidthPrescaler (TIM_TypeDef * TIMx, uint32_t PulseWidthPrescaler)

Function description

Set the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance
- **PulseWidthPrescaler:** This parameter can be one of the following values:
  - LL_TIM_PWPRSC_X1
  - LL_TIM_PWPRSC_X2
  - LL_TIM_PWPRSC_X4
  - LL_TIM_PWPRSC_X8
  - LL_TIM_PWPRSC_X16
  - LL_TIM_PWPRSC_X32
  - LL_TIM_PWPRSC_X64
  - LL_TIM_PWPRSC_X128

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_SetPulseWidthPrescaler

LL_TIM_OC_GetPulseWidthPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidthPrescaler (TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_PWPRSC_X1
  - LL_TIM_PWPRSC_X2
  - LL_TIM_PWPRSC_X4
  - LL_TIM_PWPRSC_X8
  - LL_TIM_PWPRSC_X16
  - LL_TIM_PWPRSC_X32
  - LL_TIM_PWPRSC_X64
  - LL_TIM_PWPRSC_X128

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_GetPulseWidthPrescaler

LL_TIM_OC_SetPulseWidth

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidth (TIM_TypeDef * TIMx, uint32_t PulseWidth)

Function description

Set the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance
- **PulseWidth:** This parameter can be between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_SetPulseWidth

LL_TIM_OC_GetPulseWidth

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidth (TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_GetPulseWidth

LL_TIM_IC_Config

Function name

__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC
  - LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8
  - LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8
  - LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
- CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name

__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description

Set the active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_GetActiveInput
- CCMR1 CC2S LL_TIM_IC_GetActiveInput
- CCMR2 CC3S LL_TIM_IC_GetActiveInput
- CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)

Function description

Set the prescaler of input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_SetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current prescaler value acting on an input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_GetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)

Function description

Set the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICFilter:** This parameter can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_SetFilter
- CCMR1 IC2F LL_TIM_IC_SetFilter
- CCMR2 IC3F LL_TIM_IC_SetFilter
- CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_GetFilter
- CCMR1 IC2F LL_TIM_IC_GetFilter
- CCMR2 IC3F LL_TIM_IC_GetFilter
- CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)

Function description

Set the input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPolarity:** This parameter can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_SetPolarity
- CCER CC1NP LL_TIM_IC_SetPolarity
- CCER CC2P LL_TIM_IC_SetPolarity
- CCER CC2NP LL_TIM_IC_SetPolarity
- CCER CC3P LL_TIM_IC_SetPolarity
- CCER CC3NP LL_TIM_IC_SetPolarity
- CCER CC4P LL_TIM_IC_SetPolarity
- CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_GetPolarity
- CCER CC1NP LL_TIM_IC_GetPolarity
- CCER CC2P LL_TIM_IC_GetPolarity
- CCER CC2NP LL_TIM_IC_GetPolarity
- CCER CC3P LL_TIM_IC_GetPolarity
- CCER CC3NP LL_TIM_IC_GetPolarity
- CCER CC4P LL_TIM_IC_GetPolarity
- CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)

Function description

Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)

Function description

Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (TIM_TypeDef * TIMx)

Function description

Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name

__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)

Function description

Enable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name

__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)

Function description

Disable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (TIM_TypeDef * TIMx)

Function description

Indicate whether external clock mode 2 is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name

__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)

Function description

Set the clock source of the counter clock.

Parameters

- **TIMx:** Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_TIM_CLOCKSOURCE_INTERNAL
  - LL_TIM_CLOCKSOURCE_EXT_MODE1
  - LL_TIM_CLOCKSOURCE_EXT_MODE2

Return values

- **None:**

Notes

- when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetClockSource
- SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)

Function description

Set the encoder interface mode.

Parameters

- **TIMx:** Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_TIM_ENCODERMODE_X2_TI1
  - LL_TIM_ENCODERMODE_X2_TI2
  - LL_TIM_ENCODERMODE_X4_TI12
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12
  - LL_TIM_ENCODERMODE_X1_TI1
  - LL_TIM_ENCODERMODE_X1_TI2

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)

Function description

Set the trigger output (TRGO) used for timer synchronization .

Parameters

- **TIMx:** Timer instance
- **TimerSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO_RESET
  - LL_TIM_TRGO_ENABLE
  - LL_TIM_TRGO_UPDATE
  - LL_TIM_TRGO_CC1IF
  - LL_TIM_TRGO_OC1REF
  - LL_TIM_TRGO_OC2REF
  - LL_TIM_TRGO_OC3REF
  - LL_TIM_TRGO_OC4REF
  - LL_TIM_TRGO_ENCODERCLK

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.

Reference Manual to LL API cross reference:

- CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetTriggerOutput2

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput2 (TIM_TypeDef * TIMx, uint32_t ADCSynchronization)

Function description

Set the trigger output 2 (TRGO2) used for ADC synchronization .

Parameters

- **TIMx:** Timer Instance
- **ADCSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO2_RESET
  - LL_TIM_TRGO2_ENABLE
  - LL_TIM_TRGO2_UPDATE
  - LL_TIM_TRGO2_CC1F
  - LL_TIM_TRGO2_OC1
  - LL_TIM_TRGO2_OC2
  - LL_TIM_TRGO2_OC3
  - LL_TIM_TRGO2_OC4
  - LL_TIM_TRGO2_OC5
  - LL_TIM_TRGO2_OC6
  - LL_TIM_TRGO2_OC4_RISINGFALLING
  - LL_TIM_TRGO2_OC6_RISINGFALLING
  - LL_TIM_TRGO2_OC4_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC4_RISING_OC6_FALLING
  - LL_TIM_TRGO2_OC5_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

Return values

- **None:**

Notes

- Macro IS_TIM_TRGO2_INSTANCE(TIMx) can be used to check whether or not a timer instance can be used for ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 MMS2 LL_TIM_SetTriggerOutput2

LL_TIM_SetSlaveMode

Function name

__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)

Function description

Set the synchronization mode of a slave timer.

Parameters

- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  - LL_TIM_SLAVEMODE_DISABLED
  - LL_TIM_SLAVEMODE_RESET
  - LL_TIM_SLAVEMODE_GATED
  - LL_TIM_SLAVEMODE_TRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)

Function description

Set the selects the trigger input to be used to synchronize the counter.

Parameters

- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  - LL_TIM_TS_ITR0
  - LL_TIM_TS_ITR1
  - LL_TIM_TS_ITR2
  - LL_TIM_TS_ITR3
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF
  - LL_TIM_TS_ITR4
  - LL_TIM_TS_ITR5
  - LL_TIM_TS_ITR6
  - LL_TIM_TS_ITR7
  - LL_TIM_TS_ITR8
  - LL_TIM_TS_ITR9
  - LL_TIM_TS_ITR10
  - LL_TIM_TS_ITR11

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Enable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_EnableMasterSlaveMode

LL_TIM_DisableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Disable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Indicates whether the Master/Slave mode is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name

__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)

Function description

Configure the external trigger (ETR) input.

Parameters

- **TIMx:** Timer instance
- **ETRPolarity:** This parameter can be one of the following values:
  - LL_TIM_ETR_POLARITY_NONINVERTED
  - LL_TIM_ETR_POLARITY_INVERTED
- **ETRPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ETR_PRESCALER_DIV1
  - LL_TIM_ETR_PRESCALER_DIV2
  - LL_TIM_ETR_PRESCALER_DIV4
  - LL_TIM_ETR_PRESCALER_DIV8
- **ETRFilter:** This parameter can be one of the following values:
  - LL_TIM_ETR_FILTER_FDIV1
  - LL_TIM_ETR_FILTER_FDIV1_N2
  - LL_TIM_ETR_FILTER_FDIV1_N4
  - LL_TIM_ETR_FILTER_FDIV1_N8
  - LL_TIM_ETR_FILTER_FDIV2_N6
  - LL_TIM_ETR_FILTER_FDIV2_N8
  - LL_TIM_ETR_FILTER_FDIV4_N6
  - LL_TIM_ETR_FILTER_FDIV4_N8
  - LL_TIM_ETR_FILTER_FDIV8_N6
  - LL_TIM_ETR_FILTER_FDIV8_N8
  - LL_TIM_ETR_FILTER_FDIV16_N5
  - LL_TIM_ETR_FILTER_FDIV16_N6
  - LL_TIM_ETR_FILTER_FDIV16_N8
  - LL_TIM_ETR_FILTER_FDIV32_N5
  - LL_TIM_ETR_FILTER_FDIV32_N6
  - LL_TIM_ETR_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:

- SMCR ETP LL_TIM_ConfigETR
- SMCR ETPS LL_TIM_ConfigETR
- SMCR ETF LL_TIM_ConfigETR

LL_TIM_SetETRSource

Function name

__STATIC_INLINE void LL_TIM_SetETRSource (TIM_TypeDef * TIMx, uint32_t ETRSource)

Function description

Select the external trigger (ETR) input source.

Parameters

- **TIMx:** Timer instance
- **ETRSource:** This parameter can be one of the following values:

Return values

- **None:**

Notes

- Macro IS_TIM_ETRSEL_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ETR source selection.

Reference Manual to LL API cross reference:

- AF1 ETRSEL LL_TIM_SetETRSource

LL_TIM_EnableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_EnableSMSPreload (TIM_TypeDef * TIMx)

Function description

Enable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_EnableSMSPreload

LL_TIM_DisableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_DisableSMSPreload (TIM_TypeDef * TIMx)

Function description

Disable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_DisableSMSPreload

LL_TIM_IsEnabledSMSPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledSMSPreload (TIM_TypeDef * TIMx)

Function description

Indicate whether SMS preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_IsEnabledSMSPreload

LL_TIM_SetSMSPreloadSource

Function name

__STATIC_INLINE void LL_TIM_SetSMSPreloadSource (TIM_TypeDef * TIMx, uint32_t PreloadSource)

Function description

Set the preload source of SMS.

Parameters

- **TIMx:** Timer instance
- **PreloadSource:** This parameter can be one of the following values:
  - LL_TIM_SMSPS_TIMUPDATE
  - LL_TIM_SMSPS_INDEX

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_SetSMSPreloadSource
-

LL_TIM_GetSMSPreloadSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetSMSPreloadSource (TIM_TypeDef * TIMx)

Function description

Get the preload source of SMS.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_SMSPS_TIMUPDATE
  - LL_TIM_SMSPS_INDEX

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_GetSMSPreloadSource
-

LL_TIM_EnableBRK

Function name

__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)

Function description

Enable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name

__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)

Function description

Disable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_DisableBRK

LL_TIM_ConfigBRK

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity, uint32_t BreakFilter, uint32_t BreakAFMode)

Function description

Configure the break input.

Parameters

- **TIMx:** Timer instance
- **BreakPolarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK_POLARITY_LOW
  - LL_TIM_BREAK_POLARITY_HIGH
- **BreakFilter:** This parameter can be one of the following values:
  - LL_TIM_BREAK_FILTER_FDIV1
  - LL_TIM_BREAK_FILTER_FDIV1_N2
  - LL_TIM_BREAK_FILTER_FDIV1_N4
  - LL_TIM_BREAK_FILTER_FDIV1_N8
  - LL_TIM_BREAK_FILTER_FDIV2_N6
  - LL_TIM_BREAK_FILTER_FDIV2_N8
  - LL_TIM_BREAK_FILTER_FDIV4_N6
  - LL_TIM_BREAK_FILTER_FDIV4_N8
  - LL_TIM_BREAK_FILTER_FDIV8_N6
  - LL_TIM_BREAK_FILTER_FDIV8_N8
  - LL_TIM_BREAK_FILTER_FDIV16_N5
  - LL_TIM_BREAK_FILTER_FDIV16_N6
  - LL_TIM_BREAK_FILTER_FDIV16_N8
  - LL_TIM_BREAK_FILTER_FDIV32_N5
  - LL_TIM_BREAK_FILTER_FDIV32_N6
  - LL_TIM_BREAK_FILTER_FDIV32_N8
- **BreakAFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK_AFMODE_INPUT
  - LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BKBID bit set), the Break input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, BreakAFMode must be set to LL_TIM_BREAK_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BKP LL_TIM_ConfigBRK
- BDTR BKF LL_TIM_ConfigBRK
- BDTR BKBID LL_TIM_ConfigBRK

LL_TIM_DisarmBRK

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK (TIM_TypeDef * TIMx)

Function description

Disarm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_DisarmBRK

LL_TIM_ReArmBRK

Function name

__STATIC_INLINE void LL_TIM_ReArmBRK (TIM_TypeDef * TIMx)

Function description

Re-arm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The Break input is automatically armed as soon as MOE bit is set.

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_ReArmBRK

LL_TIM_EnableBRK2

Function name

__STATIC_INLINE void LL_TIM_EnableBRK2 (TIM_TypeDef * TIMx)

Function description

Enable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_EnableBRK2

LL_TIM_DisableBRK2

Function name

__STATIC_INLINE void LL_TIM_DisableBRK2 (TIM_TypeDef * TIMx)

Function description

Disable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_DisableBRK2

LL_TIM_ConfigBRK2

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK2 (TIM_TypeDef * TIMx, uint32_t Break2Polarity, uint32_t Break2Filter, uint32_t Break2AFMode)

Function description

Configure the break 2 input.

Parameters

- **TIMx:** Timer instance
- **Break2Polarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_POLARITY_LOW
  - LL_TIM_BREAK2_POLARITY_HIGH
- **Break2Filter:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_FILTER_FDIV1
  - LL_TIM_BREAK2_FILTER_FDIV1_N2
  - LL_TIM_BREAK2_FILTER_FDIV1_N4
  - LL_TIM_BREAK2_FILTER_FDIV1_N8
  - LL_TIM_BREAK2_FILTER_FDIV2_N6
  - LL_TIM_BREAK2_FILTER_FDIV2_N8
  - LL_TIM_BREAK2_FILTER_FDIV4_N6
  - LL_TIM_BREAK2_FILTER_FDIV4_N8
  - LL_TIM_BREAK2_FILTER_FDIV8_N6
  - LL_TIM_BREAK2_FILTER_FDIV8_N8
  - LL_TIM_BREAK2_FILTER_FDIV16_N5
  - LL_TIM_BREAK2_FILTER_FDIV16_N6
  - LL_TIM_BREAK2_FILTER_FDIV16_N8
  - LL_TIM_BREAK2_FILTER_FDIV32_N5
  - LL_TIM_BREAK2_FILTER_FDIV32_N6
  - LL_TIM_BREAK2_FILTER_FDIV32_N8
- **Break2AFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_AFMODE_INPUT
  - LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BK2BID bit set), the Break 2 input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break 2 input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, Break2AFMode must be set to LL_TIM_BREAK2_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BK2P LL_TIM_ConfigBRK2
- BDTR BK2F LL_TIM_ConfigBRK2
- BDTR BK2BID LL_TIM_ConfigBRK2

LL_TIM_DisarmBRK2

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK2 (TIM_TypeDef * TIMx)

Function description

Disarm the break 2 input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break 2 input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output.

Reference Manual to LL API cross reference:

- BDTR BK2DSRM LL_TIM_DisarmBRK2

LL_TIM_ReArmBRK2

Function name

__STATIC_INLINE void LL_TIM_ReArmBRK2 (TIM_TypeDef * TIMx)

Function description

Re-arm the break 2 input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The Break 2 input is automatically armed as soon as MOE bit is set.

Reference Manual to LL API cross reference:

- BDTR BK2DSRM LL_TIM_ReArmBRK2

LL_TIM_SetOffStates

Function name

__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)

Function description

Select the outputs off state (enabled v.s.

Parameters

- **TIMx:** Timer instance
- **OffStateIdle:** This parameter can be one of the following values:
  - LL_TIM_OSSI_DISABLE
  - LL_TIM_OSSI_ENABLE
- **OffStateRun:** This parameter can be one of the following values:
  - LL_TIM_OSSR_DISABLE
  - LL_TIM_OSSR_ENABLE

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR OSSI LL_TIM_SetOffStates
- BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Enable automatic output (MOE can be set by software or automatically when a break input is active).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Disable automatic output (MOE can be set only by software).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Indicate whether automatic output is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)

Function description

Enable the outputs (set the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)

Function description

Disable the outputs (reset the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (TIM_TypeDef * TIMx)

Function description

Indicates whether outputs are enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_EnableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_EnableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Enable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1
  - LL_TIM_BKIN_SOURCE_BKCOMP2
  - LL_TIM_BKIN_SOURCE_BKCOMP3
  - LL_TIM_BKIN_SOURCE_BKCOMP4
  - LL_TIM_BKIN_SOURCE_BKCOMP5 (*)
  - LL_TIM_BKIN_SOURCE_BKCOMP6 (*)
  - LL_TIM_BKIN_SOURCE_BKCOMP7 (*)

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_EnableBreakInputSource
- AF1 BKCMP1E LL_TIM_EnableBreakInputSource
- AF1 BKCMP2E LL_TIM_EnableBreakInputSource
- AF1 BKCMP3E LL_TIM_EnableBreakInputSource
- AF1 BKCMP4E LL_TIM_EnableBreakInputSource
- AF1 BKCMP5E LL_TIM_EnableBreakInputSource
- AF1 BKCMP6E LL_TIM_EnableBreakInputSource
- AF1 BKCMP7E LL_TIM_EnableBreakInputSource
- AF2 BK2NE LL_TIM_EnableBreakInputSource
- AF2 BK2CMP1E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP2E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP3E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP4E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP5E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP6E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP7E LL_TIM_EnableBreakInputSource

LL_TIM_DisableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_DisableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Disable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1
  - LL_TIM_BKIN_SOURCE_BKCOMP2
  - LL_TIM_BKIN_SOURCE_BKCOMP3
  - LL_TIM_BKIN_SOURCE_BKCOMP4
  - LL_TIM_BKIN_SOURCE_BKCOMP5 (*)
  - LL_TIM_BKIN_SOURCE_BKCOMP6 (*)
  - LL_TIM_BKIN_SOURCE_BKCOMP7 (*)

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_DisableBreakInputSource
- AF1 BKCMP1E LL_TIM_DisableBreakInputSource
- AF1 BKCMP2E LL_TIM_DisableBreakInputSource
- AF1 BKCMP3E LL_TIM_DisableBreakInputSource
- AF1 BKCMP4E LL_TIM_DisableBreakInputSource
- AF1 BKCMP5E LL_TIM_DisableBreakInputSource
- AF1 BKCMP6E LL_TIM_DisableBreakInputSource
- AF1 BKCMP7E LL_TIM_DisableBreakInputSource
- AF2 BKINE LL_TIM_DisableBreakInputSource
- AF2 BKCMP1E LL_TIM_DisableBreakInputSource
- AF2 BKCMP2E LL_TIM_DisableBreakInputSource
- AF2 BKCMP3E LL_TIM_DisableBreakInputSource
- AF2 BKCMP4E LL_TIM_DisableBreakInputSource
- AF2 BKCMP5E LL_TIM_DisableBreakInputSource
- AF2 BKCMP6E LL_TIM_DisableBreakInputSource
- AF2 BKCMP7E LL_TIM_DisableBreakInputSource

LL_TIM_SetBreakInputSourcePolarity

Function name

__STATIC_INLINE void LL_TIM_SetBreakInputSourcePolarity (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source, uint32_t Polarity)

Function description

Set the polarity of the break signal for the timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1
  - LL_TIM_BKIN_SOURCE_BKCOMP2
  - LL_TIM_BKIN_SOURCE_BKCOMP3
  - LL_TIM_BKIN_SOURCE_BKCOMP4
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_BKIN_POLARITY_LOW
  - LL_TIM_BKIN_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINP LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP1P LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP2P LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP3P LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP4P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2INP LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP1P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP2P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP3P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP4P LL_TIM_SetBreakInputSourcePolarity

LL_TIM_EnableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_EnableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Enable asymmetrical deadtime.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_EnableAsymmetricalDeadTime

LL_TIM_DisableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_DisableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Disable asymmetrical dead-time.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_DisableAsymmetricalDeadTime

LL_TIM_IsEnabledAsymmetricalDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Indicates whether asymmetrical deadtime is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_IsEnabledAsymmetricalDeadTime

LL_TIM_SetFallingDeadTime

Function name

__STATIC_INLINE void LL_TIM_SetFallingDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the falling egde dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_SetFallingDeadTime

LL_TIM_GetFallingDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_GetFallingDeadTime (TIM_TypeDef * TIMx)

Function description

Get the falling egde dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_GetFallingDeadTime

LL_TIM_EnableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_EnableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Enable deadtime preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_EnableDeadTimePreload

LL_TIM_DisableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_DisableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Disable dead-time preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_DisableDeadTimePreload

LL_TIM_IsEnabledDeadTimePreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Indicates whether deadtime preload is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_IsEnabledDeadTimePreload

LL_TIM_ConfigDMABurst

Function name

__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength)

Function description

Configures the timer DMA burst feature.

Parameters

- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER
  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
  - LL_TIM_DMABURST_BASEADDR_CCMR3
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_DTR2
  - LL_TIM_DMABURST_BASEADDR_ECR
  - LL_TIM_DMABURST_BASEADDR_TISEL
  - LL_TIM_DMABURST_BASEADDR_AF1
  - LL_TIM_DMABURST_BASEADDR_AF2
  - LL_TIM_DMABURST_BASEADDR_OR

- **DMABurstLength:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_LENGTH_1TRANSFER
  - LL_TIM_DMABURST_LENGTH_2TRANSFERS
  - LL_TIM_DMABURST_LENGTH_3TRANSFERS
  - LL_TIM_DMABURST_LENGTH_4TRANSFERS
  - LL_TIM_DMABURST_LENGTH_5TRANSFERS
  - LL_TIM_DMABURST_LENGTH_6TRANSFERS
  - LL_TIM_DMABURST_LENGTH_7TRANSFERS
  - LL_TIM_DMABURST_LENGTH_8TRANSFERS
  - LL_TIM_DMABURST_LENGTH_9TRANSFERS
  - LL_TIM_DMABURST_LENGTH_10TRANSFERS
  - LL_TIM_DMABURST_LENGTH_11TRANSFERS
  - LL_TIM_DMABURST_LENGTH_12TRANSFERS
  - LL_TIM_DMABURST_LENGTH_13TRANSFERS
  - LL_TIM_DMABURST_LENGTH_14TRANSFERS
  - LL_TIM_DMABURST_LENGTH_15TRANSFERS
  - LL_TIM_DMABURST_LENGTH_16TRANSFERS
  - LL_TIM_DMABURST_LENGTH_17TRANSFERS
  - LL_TIM_DMABURST_LENGTH_18TRANSFERS
  - LL_TIM_DMABURST_LENGTH_19TRANSFERS
  - LL_TIM_DMABURST_LENGTH_20TRANSFERS
  - LL_TIM_DMABURST_LENGTH_21TRANSFERS
  - LL_TIM_DMABURST_LENGTH_22TRANSFERS
  - LL_TIM_DMABURST_LENGTH_23TRANSFERS
  - LL_TIM_DMABURST_LENGTH_24TRANSFERS
  - LL_TIM_DMABURST_LENGTH_25TRANSFERS
  - LL_TIM_DMABURST_LENGTH_26TRANSFERS

Return values

- **None:**

Notes

- Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.

Reference Manual to LL API cross reference:

- DCR DBL LL_TIM_ConfigDMABurst
- DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_EnableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_EnableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Enable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_EnableEncoderIndex

LL_TIM_DisableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_DisableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Disable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_DisableEncoderIndex

LL_TIM_IsEnabledEncoderIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledEncoderIndex (TIM_TypeDef * TIMx)

Function description

Indicate whether encoder index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_IsEnabledEncoderIndex

LL_TIM_SetIndexDirection

Function name

__STATIC_INLINE void LL_TIM_SetIndexDirection (TIM_TypeDef * TIMx, uint32_t IndexDirection)

Function description

Set index direction.

Parameters

- **TIMx:** Timer instance
- **IndexDirection:** This parameter can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_SetIndexDirection

LL_TIM_GetIndexDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexDirection (TIM_TypeDef * TIMx)

Function description

Get actual index direction.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_GetIndexDirection

LL_TIM_EnableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_EnableFirstIndex (TIM_TypeDef * TIMx)

Function description

Enable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_EnableFirstIndex

LL_TIM_DisableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_DisableFirstIndex (TIM_TypeDef * TIMx)

Function description

Disable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_DisableFirstIndex

LL_TIM_IsEnabledFirstIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledFirstIndex (TIM_TypeDef * TIMx)

Function description

Indicates whether first index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_IsEnabledFirstIndex

LL_TIM_SetIndexPositionning

Function name

__STATIC_INLINE void LL_TIM_SetIndexPositionning (TIM_TypeDef * TIMx, uint32_t IndexPositionning)

Function description

Set index positionning.

Parameters

- **TIMx:** Timer instance
- **IndexPositionning:** This parameter can be one of the following values:
  - LL_TIM_INDEX_POSITION_DOWN_DOWN
  - LL_TIM_INDEX_POSITION_DOWN_UP
  - LL_TIM_INDEX_POSITION_UP_DOWN
  - LL_TIM_INDEX_POSITION_UP_UP
  - LL_TIM_INDEX_POSITION_DOWN
  - LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_SetIndexPositionning

LL_TIM_GetIndexPositionning

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexPositionning (TIM_TypeDef * TIMx)

Function description

Get actual index positionning.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_POSITION_DOWN_DOWN
  - LL_TIM_INDEX_POSITION_DOWN_UP
  - LL_TIM_INDEX_POSITION_UP_DOWN
  - LL_TIM_INDEX_POSITION_UP_UP
  - LL_TIM_INDEX_POSITION_DOWN
  - LL_TIM_INDEX_POSITION_UP

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_GetIndexPositionning

LL_TIM_ConfigIDX

Function name

__STATIC_INLINE void LL_TIM_ConfigIDX (TIM_TypeDef * TIMx, uint32_t Configuration)

Function description

Configure encoder index.

Parameters

- **TIMx:** Timer instance
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_INDEX_UP or LL_TIM_INDEX_DOWN or LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_ALL or LL_TIM_INDEX_FIRST_ONLY
  - LL_TIM_INDEX_POSITION_DOWN_DOWN or ... or LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_ConfigIDX
- ECR FIDX LL_TIM_ConfigIDX
- ECR IPOS LL_TIM_ConfigIDX

LL_TIM_SetRemap

Function name

__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)

Function description

Remap TIM inputs (input channel, internal/external triggers).

Parameters

- **TIMx:** Timer instance
- **Remap:** Remap param depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of TISEL registers.

Return values

- **None:**

Notes

- Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped.

Reference Manual to LL API cross reference:

- TIM1_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI2SEL LL_TIM_SetRemap
- TIM2_TISEL TI3SEL LL_TIM_SetRemap
- TIM2_TISEL TI4SEL LL_TIM_SetRemap
- TIM3_TISEL TI1SEL LL_TIM_SetRemap
- TIM3_TISEL TI2SEL LL_TIM_SetRemap
- TIM3_TISEL TI3SEL LL_TIM_SetRemap
- TIM4_TISEL TI1SEL LL_TIM_SetRemap
- TIM4_TISEL TI2SEL LL_TIM_SetRemap
- TIM4_TISEL TI3SEL LL_TIM_SetRemap
- TIM4_TISEL TI4SEL LL_TIM_SetRemap
- TIM5_TISEL TI1SEL LL_TIM_SetRemap
- TIM5_TISEL TI2SEL LL_TIM_SetRemap
- TIM8_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI2SEL LL_TIM_SetRemap
- TIM16_TISEL TI1SEL LL_TIM_SetRemap
- TIM17_TISEL TI1SEL LL_TIM_SetRemap
- TIM20_TISEL TI1SEL LL_TIM_SetRemap

LL_TIM_EnableHSE32

Function name

__STATIC_INLINE void LL_TIM_EnableHSE32 (TIM_TypeDef * TIMx)

Function description

Enable request for HSE/32 clock used for TISEL remap.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Only TIM16 and TIM17 support HSE/32 remap

Reference Manual to LL API cross reference:

- OR HSE32EN LL_TIM_EnableHSE32

LL_TIM_DisableHSE32

Function name

__STATIC_INLINE void LL_TIM_DisableHSE32 (TIM_TypeDef * TIMx)

Function description

Disable request for HSE/32 clock used for TISEL remap.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Only TIM16 and TIM17 support HSE/32 remap

Reference Manual to LL API cross reference:

- OR HSE32EN LL_TIM_DisableHSE32

LL_TIM_IsEnabledHSE32

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledHSE32 (TIM_TypeDef * TIMx)

Function description

Indicate whether request for HSE/32 clock is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Only TIM16 and TIM17 support HSE/32 remap

Reference Manual to LL API cross reference:

- OR HSE32EN LL_TIM_IsEnabledHSE32

LL_TIM_SetOCRefClearInputSource

Function name

__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource (TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)

Function description

Set the OCREF clear input source.

Parameters

- **TIMx:** Timer instance
- **OCRefClearInputSource:** This parameter can be one of the following values:
  - LL_TIM_OCREF_CLR_INT_ETR
  - LL_TIM_OCREF_CLR_INT_COMP1
  - LL_TIM_OCREF_CLR_INT_COMP2
  - LL_TIM_OCREF_CLR_INT_COMP3
  - LL_TIM_OCREF_CLR_INT_COMP4
  - LL_TIM_OCREF_CLR_INT_COMP5 (*)
  - LL_TIM_OCREF_CLR_INT_COMP6 (*)
  - LL_TIM_OCREF_CLR_INT_COMP7 (*)

  (*) Value not defined in all devices.
-

Return values

- **None:**

Notes

- The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT
- This function can only be used in Output compare and PWM modes.
- Macro IS_TIM_OCCS_INSTANCE(TIMx) can be used to check whether or not a timer instance can configure OCREF clear input source.

Reference Manual to LL API cross reference:

- SMCR OCCS LL_TIM_SetOCRefClearInputSource
- AF2 OCRSEL LL_TIM_SetOCRefClearInputSource

LL_TIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)

Function description

Clear the update interrupt flag (UIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (TIM_TypeDef * TIMx)

Function description

Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 interrupt flag (CC1F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 interrupt flag (CC2F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 interrupt flag (CC3F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 interrupt flag (CC4F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_CC5

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC5 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 5 interrupt flag (CC5F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_ClearFlag_CC5

LL_TIM_IsActiveFlag_CC5

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC5 (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 5 interrupt flag (CC5F) is set (Capture/Compare 5 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_IsActiveFlag_CC5

LL_TIM_ClearFlag_CC6

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC6 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 6 interrupt flag (CC6F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_ClearFlag_CC6

LL_TIM_IsActiveFlag_CC6

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC6 (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 6 interrupt flag (CC6F) is set (Capture/Compare 6 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_IsActiveFlag_CC6

LL_TIM_ClearFlag_COM

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)

Function description

Clear the commutation interrupt flag (COMIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (TIM_TypeDef * TIMx)

Function description

Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)

Function description

Clear the trigger interrupt flag (TIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (TIM_TypeDef * TIMx)

Function description

Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)

Function description

Clear the break interrupt flag (BIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (TIM_TypeDef * TIMx)

Function description

Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_BRK2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK2 (TIM_TypeDef * TIMx)

Function description

Clear the break 2 interrupt flag (B2IF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_ClearFlag_BRK2

LL_TIM_IsActiveFlag_BRK2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK2 (TIM_TypeDef * TIMx)

Function description

Indicate whether break 2 interrupt flag (B2IF) is set (break 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_IsActiveFlag_BRK2

LL_TIM_ClearFlag_CC1OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_ClearFlag_SYSBRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_SYSBRK (TIM_TypeDef * TIMx)

Function description

Clear the system break interrupt flag (SBIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_ClearFlag_SYSBRK

LL_TIM_IsActiveFlag_SYSBRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_SYSBRK (TIM_TypeDef * TIMx)

Function description

Indicate whether system break interrupt flag (SBIF) is set (system break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_IsActiveFlag_SYSBRK

LL_TIM_ClearFlag_TERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TERR (TIM_TypeDef * TIMx)

Function description

Clear the transition error interrupt flag (TERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_ClearFlag_TERR

LL_TIM_IsActiveFlag_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TERR (TIM_TypeDef * TIMx)

Function description

Indicate whether transition error interrupt flag (TERRF) is set (transition error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_IsActiveFlag_TERR

LL_TIM_ClearFlag_IERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IERR (TIM_TypeDef * TIMx)

Function description

Clear the index error interrupt flag (IERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_ClearFlag_IERR

LL_TIM_IsActiveFlag_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IERR (TIM_TypeDef * TIMx)

Function description

Indicate whether index error interrupt flag (IERRF) is set (index error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_IsActiveFlag_IERR

LL_TIM_ClearFlag_DIR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_DIR (TIM_TypeDef * TIMx)

Function description

Clear the direction change interrupt flag (DIRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_ClearFlag_DIR

LL_TIM_IsActiveFlag_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_DIR (TIM_TypeDef * TIMx)

Function description

Indicate whether direction change interrupt flag (DIRF) is set (direction change interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_IsActiveFlag_DIR

LL_TIM_ClearFlag_IDX

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IDX (TIM_TypeDef * TIMx)

Function description

Clear the index interrupt flag (IDXF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_ClearFlag_IDX

LL_TIM_IsActiveFlag_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IDX (TIM_TypeDef * TIMx)

Function description

Indicate whether index interrupt flag (IDXF) is set (index interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_IsActiveFlag_IDX

LL_TIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Indicates whether the update interrupt (UIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name

__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name

__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation interrupt (COMIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)

Function description

Enable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)

Function description

Disable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (TIM_TypeDef * TIMx)

Function description

Indicates whether the break interrupt (BIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TERR (TIM_TypeDef * TIMx)

Function description

Enable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_EnableIT_TERR

LL_TIM_DisableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TERR (TIM_TypeDef * TIMx)

Function description

Disable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_DisableIT_TERR

LL_TIM_IsEnabledIT_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TERR (TIM_TypeDef * TIMx)

Function description

Indicates whether the transition error interrupt (TERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_IsEnabledIT_TERR

LL_TIM_EnableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IERR (TIM_TypeDef * TIMx)

Function description

Enable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_EnableIT_IERR

LL_TIM_DisableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IERR (TIM_TypeDef * TIMx)

Function description

Disable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_DisableIT_IERR

LL_TIM_IsEnabledIT_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IERR (TIM_TypeDef * TIMx)

Function description

Indicates whether the index error interrupt (IERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_IsEnabledIT_IERR

LL_TIM_EnableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_DIR (TIM_TypeDef * TIMx)

Function description

Enable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_EnableIT_DIR

LL_TIM_DisableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_DIR (TIM_TypeDef * TIMx)

Function description

Disable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_DisableIT_DIR

LL_TIM_IsEnabledIT_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_DIR (TIM_TypeDef * TIMx)

Function description

Indicates whether the direction change interrupt (DIRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_IsEnabledIT_DIR

LL_TIM_EnableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IDX (TIM_TypeDef * TIMx)

Function description

Enable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_EnableIT_IDX

LL_TIM_DisableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IDX (TIM_TypeDef * TIMx)

Function description

Disable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_DisableIT_IDX

LL_TIM_IsEnabledIT_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IDX (TIM_TypeDef * TIMx)

Function description

Indicates whether the index interrupt (IDXIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_IsEnabledIT_IDX

LL_TIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Indicates whether the update DMA request (UDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation DMA request (COMDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)

Function description

Generate an update event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 1 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 3 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 4 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)

Function description

Generate commutation event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)

Function description

Generate trigger event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)

Function description

Generate break event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_GenerateEvent_BRK2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK2 (TIM_TypeDef * TIMx)

Function description

Generate break 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR B2G LL_TIM_GenerateEvent_BRK2

LL_TIM_DeInit

Function name

ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)

Function description

Set TIMx registers to their reset values.

Parameters

- **TIMx:** Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name

void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Set the fields of the time base unit configuration data structure to their default values.

Parameters

- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)

Return values

- **None:**

LL_TIM_Init

Function name

ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Configure the TIMx time base unit.

Parameters

- **TIMx:** Timer Instance
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_OC_StructInit

Function name

void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Set the fields of the TIMx output channel configuration data structure to their default values.

Parameters

- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)

Return values

- **None:**

LL_TIM_OC_Init

Function name

ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Configure the TIMx output channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name

void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)

Function description

Set the fields of the TIMx input channel configuration data structure to their default values.

Parameters

- **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)

Return values

- **None:**

LL_TIM_IC_Init

Function name

ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)

Function description

Configure the TIMx input channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name

void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Fills each TIM_EncoderInitStruct field with its default value.

Parameters

- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)

Return values

- **None:**

LL_TIM_ENCODER_Init

Function name

ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Configure the encoder interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name

void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.

Parameters

- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values

- **None:**

LL_TIM_HALLSENSOR_Init

Function name

ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Configure the Hall sensor interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

Notes

- TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel
- TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.
- Channel 1 is configured as input, IC1 is mapped on TRC.
- Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.
- Channel 2 is configured in output PWM 2 mode.
- Compare value stored in TIMx_CCR2 corresponds to the commutation delay.
- OC2REF is selected as trigger output on TRGO.
- LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name

void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Set the fields of the Break and Dead Time configuration data structure to their default values.

Parameters

- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **None:**

LL_TIM_BDTR_Init

Function name

ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Configure the Break and Dead Time feature of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Break and Dead Time is initialized
  - ERROR: not applicable

Notes

- As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

90.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

90.3.1 TIM

TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI

ICx is mapped on TIx

LL_TIM_ACTIVEINPUT_INDIRECTTI

ICx is mapped on TIy

LL_TIM_ACTIVEINPUT_TRC

ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

LL_TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event

BKIN POLARITY

LL_TIM_BKIN_POLARITY_LOW

BRK BKIN input is active low

LL_TIM_BKIN_POLARITY_HIGH

BRK BKIN input is active high

BKIN SOURCE

LL_TIM_BKIN_SOURCE_BKIN

BKIN input from AF controller

LL_TIM_BKIN_SOURCE_BKCOMP1

internal signal: COMP1 output

LL_TIM_BKIN_SOURCE_BKCOMP2

internal signal: COMP2 output

LL_TIM_BKIN_SOURCE_BKCOMP3

internal signal: COMP3 output

LL_TIM_BKIN_SOURCE_BKCOMP4

internal signal: COMP4 output

LL_TIM_BKIN_SOURCE_BKCOMP5

internal signal: COMP5 output

LL_TIM_BKIN_SOURCE_BKCOMP6

internal signal: COMP6 output

LL_TIM_BKIN_SOURCE_BKCOMP7

internal signal: COMP7 output

BREAK2 AF MODE

LL_TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

Break2 Enable

LL_TIM_BREAK2_DISABLE

Break2 function disabled

LL_TIM_BREAK2_ENABLE

Break2 function enabled

BREAK2 FILTER

LL_TIM_BREAK2_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK2_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK2_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK2_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK2_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK2_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK2_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK2_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK2_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK2_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK2_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK2_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK2_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK2_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK2_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK2_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK2 POLARITY

LL_TIM_BREAK2_POLARITY_LOW

Break input BRK2 is active low

LL_TIM_BREAK2_POLARITY_HIGH

Break input BRK2 is active high

BREAK AF MODE

LL_TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

Break Enable

LL_TIM_BREAK_DISABLE

Break function disabled

LL_TIM_BREAK_ENABLE

Break function enabled

break filter

LL_TIM_BREAK_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK INPUT

LL_TIM_BREAK_INPUT_BKIN

TIMx_BKIN input

LL_TIM_BREAK_INPUT_BKIN2

TIMx_BKIN2 input

break polarity

LL_TIM_BREAK_POLARITY_LOW

Break input BRK is active low

LL_TIM_BREAK_POLARITY_HIGH

Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC

CCx DMA request sent when CCx event occurs

LL_TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY

Capture/compare control bits are updated by setting the COMG bit only

LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_TIM_CHANNEL_CH1

Timer input/output channel 1

LL_TIM_CHANNEL_CH1N

Timer complementary output channel 1

LL_TIM_CHANNEL_CH2

Timer input/output channel 2

LL_TIM_CHANNEL_CH2N

Timer complementary output channel 2

LL_TIM_CHANNEL_CH3

Timer input/output channel 3

LL_TIM_CHANNEL_CH3N

Timer complementary output channel 3

LL_TIM_CHANNEL_CH4

Timer input/output channel 4

LL_TIM_CHANNEL_CH4N

Timer complementary output channel 4

LL_TIM_CHANNEL_CH5

Timer output channel 5

LL_TIM_CHANNEL_CH6

Timer output channel 6

Clock Division

LL_TIM_CLOCKDIVISION_DIV1

tDTS=tCK_INT

LL_TIM_CLOCKDIVISION_DIV2

tDTS=2*tCK_INT

LL_TIM_CLOCKDIVISION_DIV4

tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL

The timer is clocked by the internal clock provided from the RCC

LL_TIM_CLOCKSOURCE_EXT_MODE1

Counter counts at each rising or falling edge on a selected input

LL_TIM_CLOCKSOURCE_EXT_MODE2

Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP

Timer counter counts up

LL_TIM_COUNTERDIRECTION_DOWN

Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP

Counter used as upcounter

LL_TIM_COUNTERMODE_DOWN

Counter used as downcounter

LL_TIM_COUNTERMODE_CENTER_UP

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.

LL_TIM_COUNTERMODE_CENTER_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up

LL_TIM_COUNTERMODE_CENTER_UP_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1

TIMx_CR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CR2

TIMx_CR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SMCR

TIMx_SMCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DIER

TIMx_DIER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SR

TIMx_SR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_EGR

TIMx_EGR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR1

TIMx_CCMR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR2

TIMx_CCMR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCER

TIMx_CCER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CNT

TIMx_CNT register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_PSC

TIMx_PSC register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ARR

TIMx_ARR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_RCR

TIMx_RCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR1

TIMx_CCR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR2

TIMx_CCR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR3

TIMx_CCR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR4

TIMx_CCR4 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_BDTR

TIMx_BDTR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR5

TIMx_CCR5 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR6

TIMx_CCR6 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR3

TIMx_CCMR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DTR2

TIMx_DTR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ECR

TIMx_ECR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_TISEL

TIMx_TISEL register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF1

TIMx_AF1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF2

TIMx_AF2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_OR

TIMx_OR register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER

Transfer is done to 1 register starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_2TRANSFERS

Transfer is done to 2 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_3TRANSFERS

Transfer is done to 3 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_4TRANSFERS

Transfer is done to 4 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_5TRANSFERS

Transfer is done to 5 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_6TRANSFERS

Transfer is done to 6 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_7TRANSFERS

Transfer is done to 7 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_8TRANSFERS

Transfer is done to 1 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_9TRANSFERS

Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS

Transfer is done to 10 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_11TRANSFERS

Transfer is done to 11 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_12TRANSFERS

Transfer is done to 12 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_13TRANSFERS

Transfer is done to 13 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_14TRANSFERS

Transfer is done to 14 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_15TRANSFERS

Transfer is done to 15 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_16TRANSFERS

Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS

Transfer is done to 17 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_18TRANSFERS

Transfer is done to 18 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_19TRANSFERS

Transfer is done to 19 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_20TRANSFERS

Transfer is done to 20 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_21TRANSFERS

Transfer is done to 21 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_22TRANSFERS

Transfer is done to 22 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_23TRANSFERS

Transfer is done to 23 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_24TRANSFERS

Transfer is done to 24 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_25TRANSFERS

Transfer is done to 25 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_26TRANSFERS

Transfer is done to 26 registers starting from the DMA burst base address

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1

Quadrature encoder mode 1, x2 mode - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level

LL_TIM_ENCODERMODE_X2_TI2

Quadrature encoder mode 2, x2 mode - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level

LL_TIM_ENCODERMODE_X4_TI12

Quadrature encoder mode 3, x4 mode - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction - x2 mode

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

LL_TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

LL_TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_ETR_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_ETR_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_ETR_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_ETR_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_ETR_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_ETR_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_ETR_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_ETR_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=8

LL_TIM_ETR_FILTER_FDIV8_N8

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=6

LL_TIM_ETR_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_ETR_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_ETR_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED

ETR is non-inverted, active at high level or rising edge

LL_TIM_ETR_POLARITY_INVERTED

ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1

ETR prescaler OFF

LL_TIM_ETR_PRESCALER_DIV2

ETR frequency is divided by 2

LL_TIM_ETR_PRESCALER_DIV4

ETR frequency is divided by 4

LL_TIM_ETR_PRESCALER_DIV8

ETR frequency is divided by 8

first index selection

LL_TIM_INDEX_ALL

Index is always active

LL_TIM_INDEX_FIRST_ONLY

The first Index only resets the counter

Get Flags Defines

LL_TIM_SR_UIF

Update interrupt flag

LL_TIM_SR_CC1IF

Capture/compare 1 interrupt flag

LL_TIM_SR_CC2IF

Capture/compare 2 interrupt flag

LL_TIM_SR_CC3IF

Capture/compare 3 interrupt flag

LL_TIM_SR_CC4IF

Capture/compare 4 interrupt flag

LL_TIM_SR_CC5IF

Capture/compare 5 interrupt flag

LL_TIM_SR_CC6IF

Capture/compare 6 interrupt flag

LL_TIM_SR_COMIF

COM interrupt flag

LL_TIM_SR_TIF

Trigger interrupt flag

LL_TIM_SR_BIF

Break interrupt flag

LL_TIM_SR_B2IF

Second break interrupt flag

LL_TIM_SR_CC1OF

Capture/Compare 1 overcapture flag

LL_TIM_SR_CC2OF

Capture/Compare 2 overcapture flag

LL_TIM_SR_CC3OF

Capture/Compare 3 overcapture flag

LL_TIM_SR_CC4OF

Capture/Compare 4 overcapture flag

LL_TIM_SR_SBIF

System Break interrupt flag

LL_TIM_SR_IDXF

Index interrupt flag

LL_TIM_SR_DIRF

Direction Change interrupt flag

LL_TIM_SR_IERRF

Index Error flag

LL_TIM_SR_TERRF

Transition Error flag

GROUPCH5

LL_TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

LL_TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

LL_TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

LL_TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

Clock HSE/32 request

LL_TIM_HSE_32_NOT_REQUEST

Clock HSE/32 not requested

LL_TIM_HSE_32_REQUEST

Clock HSE/32 requested for TIM16/17 TI1SEL remap

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1

No prescaler, capture is done each time an edge is detected on the capture input

LL_TIM_ICPSC_DIV2

Capture is done once every 2 events

LL_TIM_ICPSC_DIV4

Capture is done once every 4 events

LL_TIM_ICPSC_DIV8

Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_IC_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_IC_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_IC_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_IC_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_IC_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_IC_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_IC_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_IC_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_IC_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_IC_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_IC_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_IC_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_IC_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_IC_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_IC_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING

The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING

The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted

LL_TIM_IC_POLARITY_BOTHEDGE

The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

index direction selection

LL_TIM_INDEX_UP_DOWN

Index resets the counter whatever the direction

LL_TIM_INDEX_UP

Index resets the counter when up-counting only

LL_TIM_INDEX_DOWN

Index resets the counter when down-counting only

index positioning selection

LL_TIM_INDEX_POSITION_DOWN_DOWN

Index resets the counter when AB = 00

LL_TIM_INDEX_POSITION_DOWN_UP

Index resets the counter when AB = 01

LL_TIM_INDEX_POSITION_UP_DOWN

Index resets the counter when AB = 10

LL_TIM_INDEX_POSITION_UP_UP

Index resets the counter when AB = 11

LL_TIM_INDEX_POSITION_DOWN

Index resets the counter when clock is 0

LL_TIM_INDEX_POSITION_UP

Index resets the counter when clock is 1

IT Defines

LL_TIM_DIER_UIE

Update interrupt enable

LL_TIM_DIER_CC1IE

Capture/compare 1 interrupt enable

LL_TIM_DIER_CC2IE

Capture/compare 2 interrupt enable

LL_TIM_DIER_CC3IE

Capture/compare 3 interrupt enable

LL_TIM_DIER_CC4IE

Capture/compare 4 interrupt enable

LL_TIM_DIER_COMIE

COM interrupt enable

LL_TIM_DIER_TIE

Trigger interrupt enable

LL_TIM_DIER_BIE

Break interrupt enable

LL_TIM_DIER_IDXIE

Index interrupt enable

LL_TIM_DIER_DIRIE

Direction Change interrupt enable

LL_TIM_DIER_IERRIE

Index Error interrupt enable

LL_TIM_DIER_TERRIE

Transition Error interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF

LOCK OFF - No bit is write protected

LL_TIM_LOCKLEVEL_1

LOCK Level 1

LL_TIM_LOCKLEVEL_2

LOCK Level 2

LL_TIM_LOCKLEVEL_3

LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW

OCx=0 (after a dead-time if OC is implemented) when MOE=0

LL_TIM_OCIDLESTATE_HIGH

OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN

The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level

LL_TIM_OCMODE_ACTIVE

OCyREF is forced high on compare match

LL_TIM_OCMODE_INACTIVE

OCyREF is forced low on compare match

LL_TIM_OCMODE_TOGGLE

OCyREF toggles on compare match

LL_TIM_OCMODE_FORCED_INACTIVE

OCyREF is forced low

LL_TIM_OCMODE_FORCED_ACTIVE

OCyREF is forced high

LL_TIM_OCMODE_PWM1

In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.

LL_TIM_OCMODE_PWM2

In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

LL_TIM_OCMODE_RETRIG_OPM1

Retrigerrable OPM mode 1

LL_TIM_OCMODE_RETRIG_OPM2

Retrigerrable OPM mode 2

LL_TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

LL_TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

LL_TIM_OCMODE_ASSYMETRIC_PWM1

Asymmetric PWM mode 1

LL_TIM_OCMODE_ASSYMETRIC_PWM2

Asymmetric PWM mode 2

LL_TIM_OCMODE_PULSE_ON_COMPARE

Pulse on Compare mode

LL_TIM_OCMODE_DIRECTION_OUTPUT

Direction output mode

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH

OCxactive high

LL_TIM_OCPOLARITY_LOW

OCxactive low

OCREF clear input selection

LL_TIM_OCREF_CLR_INT_ETR

OCREF_CLR_INT is connected to ETRF

LL_TIM_OCREF_CLR_INT_COMP1

OCREF clear input is connected to COMP1_OUT

LL_TIM_OCREF_CLR_INT_COMP2

OCREF clear input is connected to COMP2_OUT

LL_TIM_OCREF_CLR_INT_COMP3

OCREF clear input is connected to COMP3_OUT

LL_TIM_OCREF_CLR_INT_COMP4

OCREF clear input is connected to COMP4_OUT

LL_TIM_OCREF_CLR_INT_COMP5

OCREF clear input is connected to COMP5_OUT

LL_TIM_OCREF_CLR_INT_COMP6

OCREF clear input is connected to COMP6_OUT

LL_TIM_OCREF_CLR_INT_COMP7

OCREF clear input is connected to COMP7_OUT

Output Configuration State

LL_TIM_OCSTATE_DISABLE

OCx is not active

LL_TIM_OCSTATE_ENABLE

OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE

Counter is not stopped at update event

LL_TIM_ONEPULSEMODE_REPETITIVE

Counter stops counting at the next update event

OSSI

LL_TIM_OSSI_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSI_ENABLE

When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Pulse on compare pulse width prescaler

LL_TIM_PWPRSC_X1

Pulse on compare pulse width prescaler 1

LL_TIM_PWPRSC_X2

Pulse on compare pulse width prescaler 2

LL_TIM_PWPRSC_X4

Pulse on compare pulse width prescaler 4

LL_TIM_PWPRSC_X8

Pulse on compare pulse width prescaler 8

LL_TIM_PWPRSC_X16

Pulse on compare pulse width prescaler 16

LL_TIM_PWPRSC_X32

Pulse on compare pulse width prescaler 32

LL_TIM_PWPRSC_X64

Pulse on compare pulse width prescaler 64

LL_TIM_PWPRSC_X128

Pulse on compare pulse width prescaler 128

Slave Mode

LL_TIM_SLAVEMODE_DISABLED

Slave mode disabled

LL_TIM_SLAVEMODE_RESET

Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter

LL_TIM_SLAVEMODE_GATED

Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high

LL_TIM_SLAVEMODE_TRIGGER

Trigger Mode - The counter starts at a rising edge of the trigger TRGI

LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter

LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops and is reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

SMS Preload Source

LL_TIM_SMSPS_TIMUPDATE

The SMS preload transfer is triggered by the Timer's Update event

LL_TIM_SMSPS_INDEX

The SMS preload transfer is triggered by the Index event

TIM15 Timer Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO

TIM15 input 1 is connected to GPIO

LL_TIM_TIM15_TI1_RMP_LSE

TIM15 input 1 is connected to LSE

LL_TIM_TIM15_TI1_RMP_COMP1

TIM15 input 1 is connected to COMP1_OUT

LL_TIM_TIM15_TI1_RMP_COMP2

TIM15 input 1 is connected to COMP2_OUT

LL_TIM_TIM15_TI1_RMP_COMP5

TIM15 input 1 is connected to COMP5_OUT

LL_TIM_TIM15_TI1_RMP_COMP7

TIM15 input 1 is connected to COMP7_OUT

TIM15 Timer Input Ch2 Remap

LL_TIM_TIM15_TI2_RMP_GPIO

TIM15 input 2 is connected to GPIO

LL_TIM_TIM15_TI2_RMP_COMP2

TIM15 input 2 is connected to COMP2_OUT

LL_TIM_TIM15_TI2_RMP_COMP3

TIM15 input 2 is connected to COMP3_OUT

LL_TIM_TIM15_TI2_RMP_COMP6

TIM15 input 2 is connected to COMP6_OUT

LL_TIM_TIM15_TI2_RMP_COMP7

TIM15 input 2 is connected to COMP7_OUT

TIM16 Timer Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO

TIM16 input 1 is connected to GPIO

LL_TIM_TIM16_TI1_RMP_COMP6

TIM16 input 1 is connected to COMP6_OUT

LL_TIM_TIM16_TI1_RMP_MCO

TIM16 input 1 is connected to MCO

LL_TIM_TIM16_TI1_RMP_HSE_32

TIM16 input 1 is connected to HSE/32

LL_TIM_TIM16_TI1_RMP_RTC_WK

TIM16 input 1 is connected to RTC_WAKEUP

LL_TIM_TIM16_TI1_RMP_LSE

TIM16 input 1 is connected to LSE

LL_TIM_TIM16_TI1_RMP_LSI

TIM16 input 1 is connected to LSI

TIM17 Timer Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17 input 1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_COMP5

TIM17 input 1 is connected to COMP5_OUT

LL_TIM_TIM17_TI1_RMP_MCO

TIM17 input 1 is connected to MCO

LL_TIM_TIM17_TI1_RMP_HSE_32

TIM17 input 1 is connected to HSE/32

LL_TIM_TIM17_TI1_RMP_RTC_WK

TIM17 input 1 is connected to RTC_WAKEUP

LL_TIM_TIM17_TI1_RMP_LSE

TIM17 input 1 is connected to LSE

LL_TIM_TIM17_TI1_RMP_LSI

TIM17 input 1 is connected to LSI

External Trigger Source TIM1

LL_TIM_TIM1_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM1_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM1_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM1_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM1_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM1_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM1_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM1_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD1

ADC1 analog watchdog 1

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD2

ADC1 analog watchdog 2

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD3

ADC1 analog watchdog 3

LL_TIM_TIM1_ETRSOURCE_ADC4_AWD1

ADC4 analog watchdog 1

LL_TIM_TIM1_ETRSOURCE_ADC4_AWD2

ADC4 analog watchdog 2

LL_TIM_TIM1_ETRSOURCE_ADC4_AWD3

ADC4 analog watchdog 3

TIM1 Timer Input Ch1 Remap

LL_TIM_TIM1_TI1_RMP_GPIO

TIM1 input 1 is connected to GPIO

LL_TIM_TIM1_TI1_RMP_COMP1

TIM1 input 1 is connected to COMP1_OUT

LL_TIM_TIM1_TI1_RMP_COMP2

TIM1 input 1 is connected to COMP2_OUT

LL_TIM_TIM1_TI1_RMP_COMP3

TIM1 input 1 is connected to COMP3_OUT

LL_TIM_TIM1_TI1_RMP_COMP4

TIM1 input 1 is connected to COMP4_OUT

External Trigger Source TIM20

LL_TIM_TIM20_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM20_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM20_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM20_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM20_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM20_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM20_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM20_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM20_ETRSOURCE_ADC3_AWD1

ADC3 analog watchdog 1

LL_TIM_TIM20_ETRSOURCE_ADC3_AWD2

ADC3 analog watchdog 2

LL_TIM_TIM20_ETRSOURCE_ADC3_AWD3

ADC3 analog watchdog 3

LL_TIM_TIM20_ETRSOURCE_ADC5_AWD1

ADC5 analog watchdog 1

LL_TIM_TIM20_ETRSOURCE_ADC5_AWD2

ADC5 analog watchdog 2

LL_TIM_TIM20_ETRSOURCE_ADC5_AWD3

ADC5 analog watchdog 3

TIM20 Timer Input Ch1 Remap

LL_TIM_TIM20_TI1_RMP_GPIO

TIM20 input 1 is connected to GPIO

LL_TIM_TIM20_TI1_RMP_COMP1

TIM20 input 1 is connected to COMP1_OUT

LL_TIM_TIM20_TI1_RMP_COMP2

TIM20 input 1 is connected to COMP2_OUT

LL_TIM_TIM20_TI1_RMP_COMP3

TIM20 input 1 is connected to COMP3_OUT

LL_TIM_TIM20_TI1_RMP_COMP4

TIM20 input 1 is connected to COMP4_OUT

External Trigger Source TIM2

LL_TIM_TIM2_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM2_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM2_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM2_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM2_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM2_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM2_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM2_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM2_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM2_ETRSOURCE_TIM4_ETR

ETR input is connected to TIM4 ETR

LL_TIM_TIM2_ETRSOURCE_TIM5_ETR

ETR input is connected to TIM5 ETR

LL_TIM_TIM2_ETRSOURCE_LSE

ETR input is connected to LSE

TIM2 Timer Input Ch1 Remap

LL_TIM_TIM2_TI1_RMP_GPIO

TIM2 input 1 is connected to GPIO

LL_TIM_TIM2_TI1_RMP_COMP1

TIM2 input 1 is connected to COMP1_OUT

LL_TIM_TIM2_TI1_RMP_COMP2

TIM2 input 1 is connected to COMP2_OUT

LL_TIM_TIM2_TI1_RMP_COMP3

TIM2 input 1 is connected to COMP3_OUT

LL_TIM_TIM2_TI1_RMP_COMP4

TIM2 input 1 is connected to COMP4_OUT

LL_TIM_TIM2_TI1_RMP_COMP5

TIM2 input 1 is connected to COMP5_OUT

TIM2 Timer Input Ch2 Remap

LL_TIM_TIM2_TI2_RMP_GPIO

TIM2 input 2 is connected to GPIO

LL_TIM_TIM2_TI2_RMP_COMP1

TIM2 input 2 is connected to COMP1_OUT

LL_TIM_TIM2_TI2_RMP_COMP2

TIM2 input 2 is connected to COMP2_OUT

LL_TIM_TIM2_TI2_RMP_COMP3

TIM2 input 2 is connected to COMP3_OUT

LL_TIM_TIM2_TI2_RMP_COMP4

TIM2 input 2 is connected to COMP4_OUT

LL_TIM_TIM2_TI2_RMP_COMP6

TIM2 input 2 is connected to COMP6_OUT

TIM2 Timer Input Ch3 Remap

LL_TIM_TIM2_TI3_RMP_GPIO

TIM2 input 3 is connected to GPIO

LL_TIM_TIM2_TI3_RMP_COMP4

TIM2 input 3 is connected to COMP4_OUT

TIM2 Timer Input Ch4 Remap

LL_TIM_TIM2_TI4_RMP_GPIO

TIM2 input 4 is connected to GPIO

LL_TIM_TIM2_TI4_RMP_COMP1

TIM2 input 4 is connected to COMP1_OUT

LL_TIM_TIM2_TI4_RMP_COMP2

TIM2 input 4 is connected to COMP2_OUT

External Trigger Source TIM3

LL_TIM_TIM3_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM3_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM3_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM3_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM3_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM3_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM3_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM3_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM3_ETRSOURCE_TIM2_ETR

ETR input is connected to TIM2 ETR

LL_TIM_TIM3_ETRSOURCE_TIM4_ETR

ETR input is connected to TIM4 ETR

LL_TIM_TIM3_ETRSOURCE_ADC2_AWD1

ADC2 analog watchdog 1

LL_TIM_TIM3_ETRSOURCE_ADC2_AWD2

ADC2 analog watchdog 2

LL_TIM_TIM3_ETRSOURCE_ADC2_AWD3

ADC2 analog watchdog 3

TIM3 Timer Input Ch1 Remap

LL_TIM_TIM3_TI1_RMP_GPIO

TIM3 input 1 is connected to GPIO

LL_TIM_TIM3_TI1_RMP_COMP1

TIM3 input 1 is connected to COMP1_OUT

LL_TIM_TIM3_TI1_RMP_COMP2

TIM3 input 1 is connected to COMP2_OUT

LL_TIM_TIM3_TI1_RMP_COMP3

TIM3 input 1 is connected to COMP3_OUT

LL_TIM_TIM3_TI1_RMP_COMP4

TIM3 input 1 is connected to COMP4_OUT

LL_TIM_TIM3_TI1_RMP_COMP5

TIM3 input 1 is connected to COMP5_OUT

LL_TIM_TIM3_TI1_RMP_COMP6

TIM3 input 1 is connected to COMP6_OUT

LL_TIM_TIM3_TI1_RMP_COMP7

TIM3 input 1 is connected to COMP7_OUT

TIM3 Timer Input Ch2 Remap

LL_TIM_TIM3_TI2_RMP_GPIO

TIM3 input 2 is connected to GPIO

LL_TIM_TIM3_TI2_RMP_COMP1

TIM3 input 2 is connected to COMP1_OUT

LL_TIM_TIM3_TI2_RMP_COMP2

TIM3 input 2 is connected to COMP2_OUT

LL_TIM_TIM3_TI2_RMP_COMP3

TIM3 input 2 is connected to COMP3_OUT

LL_TIM_TIM3_TI2_RMP_COMP4

TIM3 input 2 is connected to COMP4_OUT

LL_TIM_TIM3_TI2_RMP_COMP5

TIM3 input 2 is connected to COMP5_OUT

LL_TIM_TIM3_TI2_RMP_COMP6

TIM3 input 2 is connected to COMP6_OUT

LL_TIM_TIM3_TI2_RMP_COMP7

TIM3 input 2 is connected to COMP7_OUT

TIM3 Timer Input Ch3 Remap

LL_TIM_TIM3_TI3_RMP_GPIO

TIM3 input 3 is connected to GPIO

LL_TIM_TIM3_TI3_RMP_COMP3

TIM3 input 3 is connected to COMP3_OUT

External Trigger Source TIM4

LL_TIM_TIM4_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM4_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM4_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM4_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM4_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM4_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM4_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM4_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM4_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM4_ETRSOURCE_TIM5_ETR

ETR input is connected to TIM5 ETR

TIM4 Timer Input Ch1 Remap

LL_TIM_TIM4_TI1_RMP_GPIO

TIM4 input 1 is connected to GPIO

LL_TIM_TIM4_TI1_RMP_COMP1

TIM4 input 1 is connected to COMP1_OUT

LL_TIM_TIM4_TI1_RMP_COMP2

TIM4 input 1 is connected to COMP2_OUT

LL_TIM_TIM4_TI1_RMP_COMP3

TIM4 input 1 is connected to COMP3_OUT

LL_TIM_TIM4_TI1_RMP_COMP4

TIM4 input 1 is connected to COMP4_OUT

LL_TIM_TIM4_TI1_RMP_COMP5

TIM4 input 1 is connected to COMP5_OUT

LL_TIM_TIM4_TI1_RMP_COMP6

TIM4 input 1 is connected to COMP6_OUT

LL_TIM_TIM4_TI1_RMP_COMP7

TIM4 input 1 is connected to COMP7_OUT

TIM4 Timer Input Ch2 Remap

LL_TIM_TIM4_TI2_RMP_GPIO

TIM4 input 2 is connected to GPIO

LL_TIM_TIM4_TI2_RMP_COMP1

TIM4 input 2 is connected to COMP1_OUT

LL_TIM_TIM4_TI2_RMP_COMP2

TIM4 input 2 is connected to COMP2_OUT

LL_TIM_TIM4_TI2_RMP_COMP3

TIM4 input 2 is connected to COMP3_OUT

LL_TIM_TIM4_TI2_RMP_COMP4

TIM4 input 2 is connected to COMP4_OUT

LL_TIM_TIM4_TI2_RMP_COMP5

TIM4 input 2 is connected to COMP5_OUT

LL_TIM_TIM4_TI2_RMP_COMP6

TIM4 input 2 is connected to COMP6_OUT

LL_TIM_TIM4_TI2_RMP_COMP7

TIM4 input 2 is connected to COMP7_OUT

TIM4 Timer Input Ch3 Remap

LL_TIM_TIM4_TI3_RMP_GPIO

TIM4 input 3 is connected to GPIO

LL_TIM_TIM4_TI3_RMP_COMP5

TIM4 input 3 is connected to COMP5_OUT

TIM4 Timer Input Ch4 Remap

LL_TIM_TIM4_TI4_RMP_GPIO

TIM4 input 4 is connected to GPIO

LL_TIM_TIM4_TI4_RMP_COMP6

TIM4 input 4 is connected to COMP6_OUT

External Trigger Source TIM5

LL_TIM_TIM5_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM5_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM5_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM5_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM5_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM5_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM5_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM5_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM5_ETRSOURCE_TIM2_ETR

ETR input is connected to TIM2 ETR

LL_TIM_TIM5_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

TIM5 Timer Input Ch1 Remap

LL_TIM_TIM5_TI1_RMP_GPIO

TIM5 input 1 is connected to GPIO

LL_TIM_TIM5_TI1_RMP_LSI

TIM5 input 1 is connected to LSI

LL_TIM_TIM5_TI1_RMP_LSE

TIM5 input 1 is connected to LSE

LL_TIM_TIM5_TI1_RMP_RTC_WK

TIM5 input 1 is connected to RTC_WAKEUP

LL_TIM_TIM5_TI1_RMP_COMP1

TIM5 input 1 is connected to COMP1_OUT

LL_TIM_TIM5_TI1_RMP_COMP2

TIM5 input 1 is connected to COMP2_OUT

LL_TIM_TIM5_TI1_RMP_COMP3

TIM5 input 1 is connected to COMP3_OUT

LL_TIM_TIM5_TI1_RMP_COMP4

TIM5 input 1 is connected to COMP4_OUT

LL_TIM_TIM5_TI1_RMP_COMP5

TIM5 input 1 is connected to COMP5_OUT

LL_TIM_TIM5_TI1_RMP_COMP6

TIM5 input 1 is connected to COMP6_OUT

LL_TIM_TIM5_TI1_RMP_COMP7

TIM5 input 1 is connected to COMP7_OUT

TIM5 Timer Input Ch2 Remap

LL_TIM_TIM5_TI2_RMP_GPIO

TIM5 input 2 is connected to GPIO

LL_TIM_TIM5_TI2_RMP_COMP1

TIM5 input 2 is connected to COMP1_OUT

LL_TIM_TIM5_TI2_RMP_COMP2

TIM5 input 2 is connected to COMP2_OUT

LL_TIM_TIM5_TI2_RMP_COMP3

TIM5 input 2 is connected to COMP3_OUT

LL_TIM_TIM5_TI2_RMP_COMP4

TIM5 input 2 is connected to COMP4_OUT

LL_TIM_TIM5_TI2_RMP_COMP5

TIM5 input 2 is connected to COMP5_OUT

LL_TIM_TIM5_TI2_RMP_COMP6

TIM5 input 2 is connected to COMP6_OUT

LL_TIM_TIM5_TI2_RMP_COMP7

TIM5 input 2 is connected to COMP7_OUT

External Trigger Source TIM8

LL_TIM_TIM8_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM8_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM8_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM8_ETRSOURCE_COMP3

ETR input is connected to COMP3_OUT

LL_TIM_TIM8_ETRSOURCE_COMP4

ETR input is connected to COMP4_OUT

LL_TIM_TIM8_ETRSOURCE_COMP5

ETR input is connected to COMP5_OUT

LL_TIM_TIM8_ETRSOURCE_COMP6

ETR input is connected to COMP6_OUT

LL_TIM_TIM8_ETRSOURCE_COMP7

ETR input is connected to COMP7_OUT

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD1

ADC2 analog watchdog 1

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD2

ADC2 analog watchdog 2

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD3

ADC2 analog watchdog 3

LL_TIM_TIM8_ETRSOURCE_ADC3_AWD1

ADC3 analog watchdog 1

LL_TIM_TIM8_ETRSOURCE_ADC3_AWD2

ADC3 analog watchdog 2

LL_TIM_TIM8_ETRSOURCE_ADC3_AWD3

ADC3 analog watchdog 3

TIM8 Timer Input Ch1 Remap

LL_TIM_TIM8_TI1_RMP_GPIO

TIM8 input 1 is connected to GPIO

LL_TIM_TIM8_TI1_RMP_COMP1

TIM8 input 1 is connected to COMP1_OUT

LL_TIM_TIM8_TI1_RMP_COMP2

TIM8 input 1 is connected to COMP2_OUT

LL_TIM_TIM8_TI1_RMP_COMP3

TIM8 input 1 is connected to COMP3_OUT

LL_TIM_TIM8_TI1_RMP_COMP4

TIM8 input 1 is connected to COMP4_OUT

Trigger Output

LL_TIM_TRGO_RESET

UG bit from the TIMx_EGR register is used as trigger output

LL_TIM_TRGO_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output

LL_TIM_TRGO_UPDATE

Update event is used as trigger output

LL_TIM_TRGO_CC1IF

CC1 capture or a compare match is used as trigger output

LL_TIM_TRGO_OC1REF

OC1REF signal is used as trigger output

LL_TIM_TRGO_OC2REF

OC2REF signal is used as trigger output

LL_TIM_TRGO_OC3REF

OC3REF signal is used as trigger output

LL_TIM_TRGO_OC4REF

OC4REF signal is used as trigger output

LL_TIM_TRGO_ENCODERCLK

Encoder clock signal is used as trigger output

Trigger Output 2

LL_TIM_TRGO2_RESET

UG bit from the TIMx_EGR register is used as trigger output 2

LL_TIM_TRGO2_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output 2

LL_TIM_TRGO2_UPDATE

Update event is used as trigger output 2

LL_TIM_TRGO2_CC1F

CC1 capture or a compare match is used as trigger output 2

LL_TIM_TRGO2_OC1

OC1REF signal is used as trigger output 2

LL_TIM_TRGO2_OC2

OC2REF signal is used as trigger output 2

LL_TIM_TRGO2_OC3

OC3REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4

OC4REF signal is used as trigger output 2

LL_TIM_TRGO2_OC5

OC5REF signal is used as trigger output 2

LL_TIM_TRGO2_OC6

OC6REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4_RISINGFALLING

OC4REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC6_RISINGFALLING

OC6REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_RISING

OC4REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_FALLING

OC4REF rising or OC6REF falling edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_RISING

OC5REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

OC5REF rising or OC6REF falling edges are used as trigger output 2

Trigger Selection

LL_TIM_TS_ITR0

Internal Trigger 0 (ITR0) is used as trigger input

LL_TIM_TS_ITR1

Internal Trigger 1 (ITR1) is used as trigger input

LL_TIM_TS_ITR2

Internal Trigger 2 (ITR2) is used as trigger input

LL_TIM_TS_ITR3

Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED) is used as trigger input

LL_TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1) is used as trigger input

LL_TIM_TS_TI2FP2

Filtered Timer Input 2 (TI12P2) is used as trigger input

LL_TIM_TS_ETRF

Filtered external Trigger (ETRF) is used as trigger input

LL_TIM_TS_ITR4

Internal Trigger 4 (ITR4) is used as trigger input

LL_TIM_TS_ITR5

Internal Trigger 5 (ITR5) is used as trigger input

LL_TIM_TS_ITR6

Internal Trigger 6 (ITR6) is used as trigger input

LL_TIM_TS_ITR7

Internal Trigger 7 (ITR7) is used as trigger input

LL_TIM_TS_ITR8

Internal Trigger 8 (ITR8) is used as trigger input

LL_TIM_TS_ITR9

Internal Trigger 9 (ITR9) is used as trigger input

LL_TIM_TS_ITR10

Internal Trigger 10 (ITR10) is used as trigger input

LL_TIM_TS_ITR11

Internal Trigger 11 (ITR11) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR

Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request

LL_TIM_UPDATESOURCE_COUNTER

Only counter overflow/underflow generates an update request

Exported_Macros

__LL_TIM_GETFLAG_UIFCPY

Description:

- HELPER macro retrieving the UIFCPY flag from the counter value.

Parameters:

- __CNT__: Counter value

Return value:

- UIF: status bit

Notes:

- ex: __LL_TIM_GETFLAG_UIFCPY (LL_TIM_GetCounter ()); Relevant only if UIF flag remapping has been enabled (UIF status bit is copied to TIMx_CNT register bit 31)

__LL_TIM_CALC_DEADTIME

Description:

- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:

- DTG[0:7]

Notes:

- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_ARR_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR_DITHER (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_DELAY_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY_DITHER (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE_DITHER (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

Common Write and read registers Macros

LL_TIM_WriteReg

Description:

- Write a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_TIM_ReadReg

Description:

- Read a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:

- Register: value

91 LL UCPD Generic Driver

91.1 UCPD Firmware driver registers structures

91.1.1 LL_UCPD_InitTypeDef

LL_UCPD_InitTypeDef is defined in the stm32g4xx_ll_ucpd.h

Data Fields

- ***uint32_t psc_ucpdclk***
- ***uint32_t transwin***
- ***uint32_t IfrGap***
- ***uint32_t HbitClockDiv***

Field Documentation

- ***uint32_t LL_UCPD_InitTypeDef::psc_ucpdclk***

  Specify the prescaler for the UCPD clock. This parameter can be a value of ***UCPD_LL_EC_PSC***. This feature can be modified afterwards using unitary function **LL_UCPD_SetPSCClk()**.
- ***uint32_t LL_UCPD_InitTypeDef::transwin***

  Specify the number of cycles (minus 1) of the half bit clock (see HBITCLKDIV) to achieve a legal tTransitionWindow (set according to peripheral clock to define an interval of between 12 and 20 us) This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This value can be modified afterwards using unitary function **LL_UCPD_SetTransWin()**.
- ***uint32_t LL_UCPD_InitTypeDef::IfrGap***

  Specify the definition of the clock divider (minus 1) in order to generate tInterframeGap from the peripheral clock. This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This feature can be modified afterwards using unitary function **LL_UCPD_SetIfrGap()**.
- ***uint32_t LL_UCPD_InitTypeDef::HbitClockDiv***

  Specify the number of cycles (minus one) at UCPD peripheral for a half bit clock e.g. program 3 for a bit clock that takes 8 cycles of the peripheral clock "UCPD1_CLK".. This parameter can be a value between Min_Data=0x0 and Max_Data=0x3F. This feature can be modified afterwards using unitary function **LL_UCPD_SetHbitClockDiv()**.

91.2 UCPD Firmware driver API description

The following section lists the various functions of the UCPD library.

91.2.1 Detailed description of functions

LL_UCPD_Enable

Function name

__STATIC_INLINE void LL_UCPD_Enable (UCPD_TypeDef * UCPDx)

Function description

Enable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Enable

LL_UCPD_Disable

Function name

__STATIC_INLINE void LL_UCPD_Disable (UCPD_TypeDef * UCPDx)

Function description

Disable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Notes

- When disabling the UCPD, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Disable

LL_UCPD_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabled (UCPD_TypeDef const *const UCPDx)

Function description

Check if UCPD peripheral is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_IsEnabled

LL_UCPD_SetRxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrderSet (UCPD_TypeDef * UCPDx, uint32_t OrderSet)

Function description

Set the receiver ordered set detection enable.

Parameters

- **UCPDx:** UCPD Instance
- **OrderSet:** This parameter can be combination of the following values:
  - LL_UCPD_ORDERSET_SOP
  - LL_UCPD_ORDERSET_SOP1
  - LL_UCPD_ORDERSET_SOP2
  - LL_UCPD_ORDERSET_HARDRST
  - LL_UCPD_ORDERSET_CABLERST
  - LL_UCPD_ORDERSET_SOP1_DEBUG
  - LL_UCPD_ORDERSET_SOP2_DEBUG
  - LL_UCPD_ORDERSET_SOP_EXT1
  - LL_UCPD_ORDERSET_SOP_EXT2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXORDSETEN LL_UCPD_SetRxOrderSet

LL_UCPD_SetPSCClk

Function name

__STATIC_INLINE void LL_UCPD_SetPSCClk (UCPD_TypeDef * UCPDx, uint32_t Psc)

Function description

Set the prescaler for ucpd clock.

Parameters

- **UCPDx:** UCPD Instance
- **Psc:** This parameter can be one of the following values:
  - LL_UCPD_PSC_DIV1
  - LL_UCPD_PSC_DIV2
  - LL_UCPD_PSC_DIV4
  - LL_UCPD_PSC_DIV8
  - LL_UCPD_PSC_DIV16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDCLK LL_UCPD_SetPSCClk

LL_UCPD_SetTransWin

Function name

__STATIC_INLINE void LL_UCPD_SetTransWin (UCPD_TypeDef * UCPDx, uint32_t TransWin)

Function description

Set the number of cycles (minus 1) of the half bit clock.

Parameters

- **UCPDx:** UCPD Instance
- **TransWin:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TRANSWIN LL_UCPD_SetTransWin

LL_UCPD_SetIfrGap

Function name

__STATIC_INLINE void LL_UCPD_SetIfrGap (UCPD_TypeDef * UCPDx, uint32_t IfrGap)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **IfrGap:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 IFRGAP LL_UCPD_SetIfrGap

LL_UCPD_SetHbitClockDiv

Function name

__STATIC_INLINE void LL_UCPD_SetHbitClockDiv (UCPD_TypeDef * UCPDx, uint32_t HbitClock)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **HbitClock:** a value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 HBITCLKDIV LL_UCPD_SetHbitClockDiv

LL_UCPD_WakeUpEnable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpEnable (UCPD_TypeDef * UCPDx)

Function description

Enable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpEnable

LL_UCPD_WakeUpDisable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpDisable (UCPD_TypeDef * UCPDx)

Function description

Disable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpDisable

LL_UCPD_ForceClockEnable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockEnable (UCPD_TypeDef * UCPDx)

Function description

Force clock enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockEnable

LL_UCPD_ForceClockDisable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockDisable (UCPD_TypeDef * UCPDx)

Function description

Force clock disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockDisable

LL_UCPD_RxFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterEnable (UCPD_TypeDef * UCPDx)

Function description

RxFilter enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterEnable

LL_UCPD_RxFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterDisable (UCPD_TypeDef * UCPDx)

Function description

RxFilter disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterDisable

LL_UCPD_TypeCDetectionCC2Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Enable

LL_UCPD_TypeCDetectionCC2Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Disable

LL_UCPD_TypeCDetectionCC1Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Enable

LL_UCPD_TypeCDetectionCC1Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Disable

LL_UCPD_VconnDischargeEnable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeEnable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeEnable

LL_UCPD_VconnDischargeDisable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeDisable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeDisable

LL_UCPD_SignalFRSTX

Function name

__STATIC_INLINE void LL_UCPD_SignalFRSTX (UCPD_TypeDef * UCPDx)

Function description

Signal Fast Role Swap request.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSTX LL_UCPD_VconnDischargeDisable

LL_UCPD_FRSDetectionEnable

Function name

__STATIC_INLINE void LL_UCPD_FRSDetectionEnable (UCPD_TypeDef * UCPDx)

Function description

Fast Role swap RX detection enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSRXEN LL_UCPD_FRSDetectionEnable

LL_UCPD_FRSDetectionDisable

Function name

__STATIC_INLINE void LL_UCPD_FRSDetectionDisable (UCPD_TypeDef * UCPDx)

Function description

Fast Role swap RX detection disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSRXEN LL_UCPD_FRSDetectionDisable

LL_UCPD_SetccEnable

Function name

__STATIC_INLINE void LL_UCPD_SetccEnable (UCPD_TypeDef * UCPDx, uint32_t CCEnable)

Function description

Set cc enable.

Parameters

- **UCPDx:** UCPD Instance
- **CCEnable:** This parameter can be one of the following values:
  - LL_UCPD_CCENABLE_NONE
  - LL_UCPD_CCENABLE_CC1
  - LL_UCPD_CCENABLE_CC2
  - LL_UCPD_CCENABLE_CC1CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1VCONNEN LL_UCPD_SetccEnable

LL_UCPD_SetSNKRole

Function name

__STATIC_INLINE void LL_UCPD_SetSNKRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SNK role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSNKRole

LL_UCPD_SetSRCRole

Function name

__STATIC_INLINE void LL_UCPD_SetSRCRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SRC role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSRCRole

LL_UCPD_GetRole

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetRole (UCPD_TypeDef const *const UCPDx)

Function description

Get UCPD Role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_UCPD_ROLE_SNK
  - LL_UCPD_ROLE_SRC

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_GetRole

LL_UCPD_SetRpResistor

Function name

__STATIC_INLINE void LL_UCPD_SetRpResistor (UCPD_TypeDef * UCPDx, uint32_t Resistor)

Function description

Set Rp resistor.

Parameters

- **UCPDx:** UCPD Instance
- **Resistor:** This parameter can be one of the following values:
  - LL_UCPD_RESISTOR_DEFAULT
  - LL_UCPD_RESISTOR_1_5A
  - LL_UCPD_RESISTOR_3_0A
  - LL_UCPD_RESISTOR_NONE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANASUBMODE LL_UCPD_SetRpResistor

LL_UCPD_SetCCPin

Function name

__STATIC_INLINE void LL_UCPD_SetCCPin (UCPD_TypeDef * UCPDx, uint32_t CCPin)

Function description

Set CC pin.

Parameters

- **UCPDx:** UCPD Instance
- **CCPin:** This parameter can be one of the following values:
  - LL_UCPD_CCPIN_CC1
  - LL_UCPD_CCPIN_CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYCCSEL LL_UCPD_SetCCPin

LL_UCPD_RxEnable

Function name

__STATIC_INLINE void LL_UCPD_RxEnable (UCPD_TypeDef * UCPDx)

Function description

Rx enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxEnable

LL_UCPD_RxDisable

Function name

__STATIC_INLINE void LL_UCPD_RxDisable (UCPD_TypeDef * UCPDx)

Function description

Rx disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxDisable

LL_UCPD_SetRxMode

Function name

__STATIC_INLINE void LL_UCPD_SetRxMode (UCPD_TypeDef * UCPDx, uint32_t RxMode)

Function description

Set Rx mode.

Parameters

- **UCPDx:** UCPD Instance
- **RxMode:** This parameter can be one of the following values:
  - LL_UCPD_RXMODE_NORMAL
  - LL_UCPD_RXMODE_BIST_TEST_DATA

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RXMODE LL_UCPD_SetRxMode

LL_UCPD_SendHardReset

Function name

__STATIC_INLINE void LL_UCPD_SendHardReset (UCPD_TypeDef * UCPDx)

Function description

Send Hard Reset.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXHRST LL_UCPD_SendHardReset

LL_UCPD_SendMessage

Function name

__STATIC_INLINE void LL_UCPD_SendMessage (UCPD_TypeDef * UCPDx)

Function description

Send message.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXSEND LL_UCPD_SendMessage

LL_UCPD_SetTxMode

Function name

__STATIC_INLINE void LL_UCPD_SetTxMode (UCPD_TypeDef * UCPDx, uint32_t TxMode)

Function description

Set Tx mode.

Parameters

- **UCPDx:** UCPD Instance
- **TxMode:** This parameter can be one of the following values:
  - LL_UCPD_TXMODE_NORMAL
  - LL_UCPD_TXMODE_CABLE_RESET
  - LL_UCPD_TXMODE_BIST_CARRIER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXMODE LL_UCPD_SetTxMode

LL_UCPD_EnableIT_FRS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_FRS (UCPD_TypeDef * UCPDx)

Function description

Enable FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_EnableIT_FRS

LL_UCPD_EnableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_EnableIT_TypeCEventCC2

LL_UCPD_EnableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_EnableIT_TypeCEventCC1

LL_UCPD_EnableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Enable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_EnableIT_RxMsgEnd

LL_UCPD_EnableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Enable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_EnableIT_RxOvr

LL_UCPD_EnableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Enable Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_EnableIT_RxHRST

LL_UCPD_EnableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Enable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_EnableIT_RxOrderSet

LL_UCPD_EnableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Enable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_EnableIT_RxNE

LL_UCPD_EnableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Enable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_EnableIT_TxUND

LL_UCPD_EnableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_EnableIT_TxHRSTSENT

LL_UCPD_EnableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_EnableIT_TxHRSTDISC

LL_UCPD_EnableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_EnableIT_TxMSGABT

LL_UCPD_EnableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_EnableIT_TxMSGSENT

LL_UCPD_EnableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_EnableIT_TxMSGDISC

LL_UCPD_EnableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Enable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_EnableIT_TxIS

LL_UCPD_DisableIT_FRS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_FRS (UCPD_TypeDef * UCPDx)

Function description

Disable FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_DisableIT_FRS

LL_UCPD_DisableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_DisableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_DisableIT_TypeCEventCC1

LL_UCPD_DisableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Disable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_DisableIT_RxMsgEnd

LL_UCPD_DisableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Disable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_DisableIT_RxOvr

LL_UCPD_DisableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Disable Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_DisableIT_RxHRST

LL_UCPD_DisableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Disable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_DisableIT_RxOrderSet

LL_UCPD_DisableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Disable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_DisableIT_RxNE

LL_UCPD_DisableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Disable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_DisableIT_TxUND

LL_UCPD_DisableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_DisableIT_TxHRSTSENT

LL_UCPD_DisableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_DisableIT_TxHRSTDISC

LL_UCPD_DisableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_DisableIT_TxMSGABT

LL_UCPD_DisableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_DisableIT_TxMSGSENT

LL_UCPD_DisableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_DisableIT_TxMSGDISC

LL_UCPD_DisableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Disable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_DisableIT_TxIS

LL_UCPD_IsEnableIT_FRS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_FRS (UCPD_TypeDef const *const UCPDx)

Function description

Check if FRS interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_DisableIT_FRS

LL_UCPD_IsEnableIT_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_IsEnableIT_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR2 TYPECEVT1IE LL_UCPD_IsEnableIT_TypeCEventCC1

LL_UCPD_IsEnableIT_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_IsEnableIT_RxMsgEnd

LL_UCPD_IsEnableIT_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_IsEnableIT_RxOvr

LL_UCPD_IsEnableIT_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard resrt interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_IsEnableIT_RxHRST

LL_UCPD_IsEnableIT_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_IsEnableIT_RxOrderSet

LL_UCPD_IsEnableIT_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_IsEnableIT_RxNE

LL_UCPD_IsEnableIT_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_IsEnableIT_TxUND

LL_UCPD_IsEnableIT_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_IsEnableIT_TxHRSTSENT

LL_UCPD_IsEnableIT_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_IsEnableIT_TxHRSTDISC

LL_UCPD_IsEnableIT_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_IsEnableIT_TxMSGABT

LL_UCPD_IsEnableIT_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_IsEnableIT_TxMSGSENT

LL_UCPD_IsEnableIT_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_IsEnableIT_TxMSGDISC

LL_UCPD_IsEnableIT_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_IsEnableIT_TxIS

LL_UCPD_ClearFlag_FRS

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_FRS (UCPD_TypeDef * UCPDx)

Function description

Clear FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FRSEVTIE LL_UCPD_ClearFlag_FRS

LL_UCPD_ClearFlag_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT2IE LL_UCPD_ClearFlag_TypeCEventCC2

LL_UCPD_ClearFlag_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT1IE LL_UCPD_ClearFlag_TypeCEventCC1

LL_UCPD_ClearFlag_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Clear Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXMSGENDIE LL_UCPD_ClearFlag_RxMsgEnd

LL_UCPD_ClearFlag_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Clear Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXOVRIE LL_UCPD_ClearFlag_RxOvr

LL_UCPD_ClearFlag_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Clear Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXHRSTDETIE LL_UCPD_ClearFlag_RxHRST

LL_UCPD_ClearFlag_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Clear Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXORDDETIE LL_UCPD_ClearFlag_RxOrderSet

LL_UCPD_ClearFlag_TxUND

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxUND (UCPD_TypeDef * UCPDx)

Function description

Clear TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXUNDIE LL_UCPD_ClearFlag_TxUND

LL_UCPD_ClearFlag_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTSENTIE LL_UCPD_ClearFlag_TxHRSTSENT

LL_UCPD_ClearFlag_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTDISCIE LL_UCPD_ClearFlag_TxHRSTDISC

LL_UCPD_ClearFlag_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGABTIE LL_UCPD_ClearFlag_TxMSGABT

LL_UCPD_ClearFlag_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGSENTIE LL_UCPD_ClearFlag_TxMSGSENT

LL_UCPD_ClearFlag_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGDISCIE LL_UCPD_ClearFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_FRS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_FRS (UCPD_TypeDef const *const UCPDx)

Function description

Check if FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR FRSEVT LL_UCPD_IsActiveFlag_FRS

LL_UCPD_IsActiveFlag_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TYPECEVT2 LL_UCPD_IsActiveFlag_TypeCEventCC2

LL_UCPD_IsActiveFlag_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TYPECEVT1 LL_UCPD_IsActiveFlag_TypeCEventCC1

LL_UCPD_IsActiveFlag_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXMSGEND LL_UCPD_IsActiveFlag_RxMsgEnd

LL_UCPD_IsActiveFlag_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXOVR LL_UCPD_IsActiveFlag_RxOvr

LL_UCPD_IsActiveFlag_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXHRSTDET LL_UCPD_IsActiveFlag_RxHRST

LL_UCPD_IsActiveFlag_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXORDDET LL_UCPD_IsActiveFlag_RxOrderSet

LL_UCPD_IsActiveFlag_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_UCPD_IsActiveFlag_RxNE

LL_UCPD_IsActiveFlag_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXUND LL_UCPD_IsActiveFlag_TxUND

LL_UCPD_IsActiveFlag_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTSENT LL_UCPD_IsActiveFlag_TxHRSTSENT

LL_UCPD_IsActiveFlag_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTDISC LL_UCPD_IsActiveFlag_TxHRSTDISC

LL_UCPD_IsActiveFlag_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGABT LL_UCPD_IsActiveFlag_TxMSGABT

LL_UCPD_IsActiveFlag_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGSENT LL_UCPD_IsActiveFlag_TxMSGSENT

LL_UCPD_IsActiveFlag_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGDISC LL_UCPD_IsActiveFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_IsActiveFlag_TxIS

LL_UCPD_GetTypeCVstateCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC2 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC2

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC2

LL_UCPD_GetTypeCVstateCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC1 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC1

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC1

LL_UCPD_RxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_RxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMAEnable

LL_UCPD_RxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_RxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMADisable

LL_UCPD_TxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_TxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMAEnable

LL_UCPD_TxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_TxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMADisable

LL_UCPD_IsEnabledTxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledTxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Tx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_UCPD_IsEnabledTxDMA

LL_UCPD_IsEnabledRxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledRxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Rx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_UCPD_IsEnabledRxDMA

LL_UCPD_WriteTxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_WriteTxOrderSet (UCPD_TypeDef * UCPDx, uint32_t TxOrderSet)

Function description

write the orderset for Tx message

Parameters

- **UCPDx:** UCPD Instance
- **TxOrderSet:** one of the following value
  - LL_UCPD_ORDERED_SET_SOP
  - LL_UCPD_ORDERED_SET_SOP1
  - LL_UCPD_ORDERED_SET_SOP2
  - LL_UCPD_ORDERED_SET_HARD_RESET
  - LL_UCPD_ORDERED_SET_CABLE_RESET
  - LL_UCPD_ORDERED_SET_SOP1_DEBUG
  - LL_UCPD_ORDERED_SET_SOP2_DEBUG

Return values

- **None:**

Reference Manual to LL API cross reference:

- TX_ORDSET TXORDSET LL_UCPD_WriteTxOrderSet

LL_UCPD_WriteTxPaySize

Function name

__STATIC_INLINE void LL_UCPD_WriteTxPaySize (UCPD_TypeDef * UCPDx, uint32_t TxPaySize)

Function description

write the Tx paysize

Parameters

- **UCPDx:** UCPD Instance
- **TxPaySize:**

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TX_PAYSZ TXPAYSZ LL_UCPD_WriteTxPaySize

LL_UCPD_WriteData

Function name

__STATIC_INLINE void LL_UCPD_WriteData (UCPD_TypeDef * UCPDx, uint8_t Data)

Function description

Write data.

Parameters

- **UCPDx:** UCPD Instance
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TXDR DR LL_UCPD_WriteData

LL_UCPD_ReadRxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

read RX the orderset

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxOrderSet:** one of the following value
  - LL_UCPD_RXORDSET_SOP
  - LL_UCPD_RXORDSET_SOP1
  - LL_UCPD_RXORDSET_SOP2
  - LL_UCPD_RXORDSET_SOP1_DEBUG
  - LL_UCPD_RXORDSET_SOP2_DEBUG
  - LL_UCPD_RXORDSET_CABLE_RESET
  - LL_UCPD_RXORDSET_SOPEXT1
  - LL_UCPD_RXORDSET_SOPEXT2

Reference Manual to LL API cross reference:

- RX_ORDSET RXORDSET LL_UCPD_ReadRxOrderSet

LL_UCPD_ReadRxPaySize

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxPaySize (UCPD_TypeDef const *const UCPDx)

Function description

Read the Rx paysize.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RXPaysize.:**

Reference Manual to LL API cross reference:

- TX_PAYSZ TXPAYSZ LL_UCPD_ReadRxPaySize

LL_UCPD_ReadData

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadData (UCPD_TypeDef const *const UCPDx)

Function description

Read data.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TXDR RXDATA LL_UCPD_ReadData

LL_UCPD_SetRxOrdExt1

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt1 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext1.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT1 RXSOPX1 LL_UCPD_SetRxOrdExt1

LL_UCPD_SetRxOrdExt2

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt2 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext2.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT2 RXSOPX2 LL_UCPD_SetRxOrdExt2

LL_UCPD_DeInit

Function name

ErrorStatus LL_UCPD_DeInit (UCPD_TypeDef * UCPDx)

Function description

De-initialize the UCPD registers to their default reset values.

Parameters

- **UCPDx:** ucpd Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ucpd registers are de-initialized
  - ERROR: ucpd registers are not de-initialized

LL_UCPD_Init

Function name

ErrorStatus LL_UCPD_Init (UCPD_TypeDef * UCPDx, LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Initialize the ucpd registers according to the specified parameters in UCPD_InitStruct.

Parameters

- **UCPDx:** UCPD Instance
- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure that contains the configuration information for the UCPD peripheral.

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in ucpd configuration registers can only be written when the ucpd is disabled (ucpd_CR1_SPE bit =0), UCPD peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_UCPD_StructInit

Function name

void LL_UCPD_StructInit (LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Set each LL_UCPD_InitTypeDef field to default value.

Parameters

- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

91.3 UCPD Firmware driver defines

The following section lists the various define and macros of the module.

91.3.1 UCPD

UCPD

CC pin enable

LL_UCPD_CCENABLE_NONE

Neither PHY is activated (e.g. disabled state of source)

LL_UCPD_CCENABLE_CC1

Controls apply to only CC1

LL_UCPD_CCENABLE_CC2

Controls apply to only CC1

LL_UCPD_CCENABLE_CC1CC2

Controls apply to both CC1 and CC2 (normal usage for sink/source)

CC pin selection

LL_UCPD_CCPIN_CC1

Use CC1 IO for power delivery communication

LL_UCPD_CCPIN_CC2

Use CC2 IO for power delivery communication

CCx event

LL_UCPD_SNK_CC1_VOPEN

CC1 Sink Open state

LL_UCPD_SNK_CC1_VRP

CC1 Sink vRP default state

LL_UCPD_SNK_CC1_VRP15A

CC1 Sink vRP 1.5A state

LL_UCPD_SNK_CC1_VRP30A

CC1 Sink vRP 3.0A state

LL_UCPD_SNK_CC2_VOPEN

CC2 Sink Open state

LL_UCPD_SNK_CC2_VRP

CC2 Sink vRP default state

LL_UCPD_SNK_CC2_VRP15A

CC2 Sink vRP 1.5A state

LL_UCPD_SNK_CC2_VRP30A

CC2 Sink vRP 3.0A state

LL_UCPD_SRC_CC1_VRA

CC1 Source vRA state

LL_UCPD_SRC_CC1_VRD

CC1 Source vRD state

LL_UCPD_SRC_CC1_OPEN

CC1 Source Open state

LL_UCPD_SRC_CC2_VRA

CC2 Source vRA state

LL_UCPD_SRC_CC2_VRD

CC2 Source vRD state

LL_UCPD_SRC_CC2_OPEN

CC2 Source Open state

ordered set configuration

LL_UCPD_ORDERSET_SOP

SOP Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1

SOP' Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2

SOP'' Ordered set detection enabled

LL_UCPD_ORDERSET_HARDRST

Hard Reset Ordered set detection enabled

LL_UCPD_ORDERSET_CABLERST

Cable Reset Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1_DEBUG

SOP' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2_DEBUG

SOP'' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT1

SOP extension#1 Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT2

SOP extension#2 Ordered set detection enabled

Get Flags Defines

LL_UCPD_SR_TXIS

Transmit interrupt status

LL_UCPD_SR_TXMSGDISC

Transmit message discarded interrupt

LL_UCPD_SR_TXMSGSENT

Transmit message sent interrupt

LL_UCPD_SR_TXMSGABT

Transmit message abort interrupt

LL_UCPD_SR_HRSTDISC

HRST discarded interrupt

LL_UCPD_SR_HRSTSENT

HRST sent interrupt

LL_UCPD_SR_TXUND

Tx data underrun condition interrupt

LL_UCPD_SR_RXNE

Receive data register not empty interrupt

LL_UCPD_SR_RXORDDET

Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_SR_RXHRSTDET

Rx Hard Reset detect interrupt

LL_UCPD_SR_RXOVR

Rx data overflow interrupt

LL_UCPD_SR_RXMSGEND

Rx message received

LL_UCPD_SR_RXERR

Rx error

LL_UCPD_SR_TYPECEVT1

Type C voltage level event on CC1

LL_UCPD_SR_TYPECEVT2

Type C voltage level event on CC2

LL_UCPD_SR_TYPEC_VSTATE_CC1

Status of DC level on CC1 pin

LL_UCPD_SR_TYPEC_VSTATE_CC2

Status of DC level on CC2 pin

LL_UCPD_SR_FRSEVT

Fast Role Swap detection event

IT Defines

LL_UCPD_IMR_TXIS

Enable transmit interrupt status

LL_UCPD_IMR_TXMSGDISC

Enable transmit message discarded interrupt

LL_UCPD_IMR_TXMSGSENT

Enable transmit message sent interrupt

LL_UCPD_IMR_TXMSGABT

Enable transmit message abort interrupt

LL_UCPD_IMR_HRSTDISC

Enable HRST discarded interrupt

LL_UCPD_IMR_HRSTSENT

Enable HRST sent interrupt

LL_UCPD_IMR_TXUND

Enable tx data underrun condition interrupt

LL_UCPD_IMR_RXNE

Enable Receive data register not empty interrupt

LL_UCPD_IMR_RXORDDET

Enable Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_IMR_RXHRSTDET

Enable Rx Hard Reset detect interrupt

LL_UCPD_IMR_RXOVR

Enable Rx data overflow interrupt

LL_UCPD_IMR_RXMSGEND

Enable Rx message received

LL_UCPD_IMR_TYPECEVT1

Enable Type C voltage level event on CC1

LL_UCPD_IMR_TYPECEVT2

Enable Type C voltage level event on CC2

LL_UCPD_IMR_FRSEVT

Enable fast Role Swap detection event

Role Mode

LL_UCPD_ROLE_SNK

Mode SNK Rd

LL_UCPD_ROLE_SRC

Mode SRC Rp

Ordered sets value

LL_UCPD_SYNC1

K-code for Startsynch #1

LL_UCPD_SYNC2

K-code for Startsynch #2

LL_UCPD_SYNC3

K-code for Startsynch #3

LL_UCPD_RST1

K-code for Hard Reset #1

LL_UCPD_RST2

K-code for Hard Reset #2

LL_UCPD_EOP

K-code for EOP End of Packet

LL_UCPD_ORDERED_SET_SOP

SOP Ordered set coding

LL_UCPD_ORDERED_SET_SOP1

SOP' Ordered set coding

LL_UCPD_ORDERED_SET_SOP2

SOP'' Ordered set coding

LL_UCPD_ORDERED_SET_HARD_RESET

Hard Reset Ordered set coding

LL_UCPD_ORDERED_SET_CABLE_RESET

Cable Reset Ordered set coding

LL_UCPD_ORDERED_SET_SOP1_DEBUG

SOP' Debug Ordered set coding

LL_UCPD_ORDERED_SET_SOP2_DEBUG

SOP'' Debug Ordered set coding

prescaler for UCPDCLK

LL_UCPD_PSC_DIV1

Bypass pre-scaling / divide by 1

LL_UCPD_PSC_DIV2

Pre-scale clock by dividing by 2

LL_UCPD_PSC_DIV4

Pre-scale clock by dividing by 4

LL_UCPD_PSC_DIV8

Pre-scale clock by dividing by 8

LL_UCPD_PSC_DIV16

Pre-scale clock by dividing by 16

Resistor value

LL_UCPD_RESISTOR_DEFAULT

Rp default

LL_UCPD_RESISTOR_1_5A

Rp 1.5 A

LL_UCPD_RESISTOR_3_0A

Rp 3.0 A

LL_UCPD_RESISTOR_NONE

No resistor

Receiver mode

LL_UCPD_RXMODE_NORMAL

Normal receive mode

LL_UCPD_RXMODE_BIST_TEST_DATA

BIST receive mode (BIST Test Data Mode)

Rx ordered set code detected

LL_UCPD_RXORDSET_SOP

SOP code detected in receiver

LL_UCPD_RXORDSET_SOP1

SOP' code detected in receiver

LL_UCPD_RXORDSET_SOP2

SOP'' code detected in receiver

LL_UCPD_RXORDSET_SOP1_DEBUG

SOP' Debug code detected in receiver

LL_UCPD_RXORDSET_SOP2_DEBUG

SOP'' Debug code detected in receiver

LL_UCPD_RXORDSET_CABLE_RESET

Cable Reset code detected in receiver

LL_UCPD_RXORDSET_SOPEXT1

SOP extension#1 code detected in receiver

LL_UCPD_RXORDSET_SOPEXT2

SOP extension#2 code detected in receiver

Type of Tx packet

LL_UCPD_TXMODE_NORMAL

Initiate the transfer of a Tx message

LL_UCPD_TXMODE_CABLE_RESET

Trigger a the transfer of a Cable Reset sequence

LL_UCPD_TXMODE_BIST_CARRIER2

Trigger a BIST test sequence send (BIST Carrier Mode 2)

Common Write and read registers Macros

LL_UCPD_WriteReg

Description:

- Write a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_UCPD_ReadReg

Description:

- Read a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be read

Return value:

- Register: value

92 LL USART Generic Driver

92.1 USART Firmware driver registers structures

92.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32g4xx_ll_usart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::PrescalerValue***

  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***USART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_USART_SetPrescaler()**.
- ***uint32_t LL_USART_InitTypeDef::BaudRate***

  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***

  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***

  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.
- ***uint32_t LL_USART_InitTypeDef::OverSampling***

  Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of ***USART_LL_EC_OVERSAMPLING***.This feature can be modified afterwards using unitary function **LL_USART_SetOverSampling()**.

92.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32g4xx_ll_usart.h

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t LastBitClockPulse***

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***

  Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_LL_EC_CLOCK***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***

  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_LL_EC_POLARITY***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***

  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_LL_EC_PHASE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***

  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_LL_EC_LASTCLKPULSE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

92.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

92.2.1 Detailed description of functions

LL_USART_Enable

Function name

__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)

Function description

USART Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Enable

LL_USART_Disable

Function name

__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)

Function description

USART Disable (all USART prescalers and outputs are disabled)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_ISR are set to their default values.

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled (USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_IsEnabled

LL_USART_EnableFIFO

Function name

__STATIC_INLINE void LL_USART_EnableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_EnableFIFO

LL_USART_DisableFIFO

Function name

__STATIC_INLINE void LL_USART_DisableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_DisableFIFO

LL_USART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledFIFO (USART_TypeDef * USARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_IsEnabledFIFO

LL_USART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetTXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_SetTXFIFOThreshold

LL_USART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXFIFOThreshold (USART_TypeDef * USARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_GetTXFIFOThreshold

LL_USART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetRXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_SetRXFIFOThreshold

LL_USART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXFIFOThreshold (USART_TypeDef * USARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_GetRXFIFOThreshold

LL_USART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_USART_ConfigFIFOsThreshold (USART_TypeDef * USARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **USARTx:** USART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_USART_ConfigFIFOsThreshold

LL_USART_EnableInStopMode

Function name

__STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)

Function description

USART enabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC.
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_EnableInStopMode

LL_USART_DisableInStopMode

Function name

__STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)

Function description

USART disabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is disabled, USART is not able to wake up the MCU from Stop mode
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_DisableInStopMode

LL_USART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name

__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_SetTransferDirection
- CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (USART_TypeDef * USARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_GetTransferDirection
- CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name

__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled).

Parameters

- **USARTx:** USART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_SetParity
- CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name

__STATIC_INLINE uint32_t LL_USART_GetParity (USART_TypeDef * USARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_GetParity
- CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance
- **Method:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (USART_TypeDef * USARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name

__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)

Function description

Set Word length (i.e.

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_SetDataWidth
- CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_USART_GetDataWidth (USART_TypeDef * USARTx)

Function description

Return Word length (i.e.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_GetDataWidth
- CR1 M1 LL_USART_GetDataWidth

LL_USART_EnableMuteMode

Function name

__STATIC_INLINE void LL_USART_EnableMuteMode (USART_TypeDef * USARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_EnableMuteMode

LL_USART_DisableMuteMode

Function name

__STATIC_INLINE void LL_USART_DisableMuteMode (USART_TypeDef * USARTx)

Function description

Prevent Mute Mode use.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_DisableMuteMode

LL_USART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode (USART_TypeDef * USARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_IsEnabledMuteMode

LL_USART_SetOverSampling

Function name

__STATIC_INLINE void LL_USART_SetOverSampling (USART_TypeDef * USARTx, uint32_t OverSampling)

Function description

Set Oversampling to 8-bit or 16-bit mode.

Parameters

- **USARTx:** USART Instance
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name

__STATIC_INLINE uint32_t LL_USART_GetOverSampling (USART_TypeDef * USARTx)

Function description

Return Oversampling mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name

__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)

Function description

Configure if Clock pulse of the last data bit is output to the SCLK pin or not.

Parameters

- **USARTx:** USART Instance
- **LastBitClockPulse:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name

__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (USART_TypeDef * USARTx)

Function description

Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name

__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)

Function description

Select the phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPhase (USART_TypeDef * USARTx)

Function description

Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name

__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description

Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (USART_TypeDef * USARTx)

Function description

Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name

__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description

Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters

- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_SetPrescaler

Function name

__STATIC_INLINE void LL_USART_SetPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_SetPrescaler

LL_USART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetPrescaler (USART_TypeDef * USARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_GetPrescaler

LL_USART_EnableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)

Function description

Enable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)

Function description

Disable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (USART_TypeDef * USARTx)

Function description

Indicate if Clock output on SCLK pin is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **USARTx:** USART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (USART_TypeDef * USARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name

__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_ConfigCharacter
- CR1 PCE LL_USART_ConfigCharacter
- CR1 M0 LL_USART_ConfigCharacter
- CR1 M1 LL_USART_ConfigCharacter
- CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **USARTx:** USART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (USART_TypeDef * USARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (USART_TypeDef * USARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (USART_TypeDef * USARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **USARTx:** USART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (USART_TypeDef * USARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_SetTransferBitOrder

LL_USART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (USART_TypeDef * USARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Enable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Disable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_DisableAutoBaudRate

LL_USART_IsEnabledAutoBaud

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (USART_TypeDef * USARTx)

Function description

Indicate if Auto Baud-Rate Detection mechanism is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name

__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)

Function description

Set Auto Baud-Rate mode bits.

Parameters

- **USARTx:** USART Instance
- **AutoBaudRateMode:** This parameter can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (USART_TypeDef * USARTx)

Function description

Return Auto Baud-Rate mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name

__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name

__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (USART_TypeDef * USARTx)

Function description

Indicate if Receiver Timeout feature is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the USART node.

Parameters

- **USARTx:** USART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the USART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_ConfigNodeAddress
- CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (USART_TypeDef * USARTx)

Function description

Return 8 bit Address of the USART node as set in ADD field of CR2.

Parameters

- **USARTx:** USART Instance

Return values

- **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (USART_TypeDef * USARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **USARTx:** USART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_SetHWFlowCtrl
- CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_GetHWFlowCtrl
- CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)

Function description

Enable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_EnableOneBitSamp

LL_USART_DisableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)

Function description

Disable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (USART_TypeDef * USARTx)

Function description

Indicate if One bit sampling method is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)

Function description

Enable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)

Function description

Disable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (USART_TypeDef * USARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetWKUPType

Function name

__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance
- **Type:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_USART_GetWKUPType (USART_TypeDef * USARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_GetWKUPType

LL_USART_SetBaudRate

Function name

__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling, uint32_t BaudRate)

Function description

Configure USART BRR register for achieving expected Baud Rate value.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values
- Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_USART_GetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling)

Function description

Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name

__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)

Function description

Set Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance
- **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (USART_TypeDef * USARTx)

Function description

Get Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name

__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)

Function description

Set Block Length value in reception.

Parameters

- **USARTx:** USART Instance
- **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetBlockLength (USART_TypeDef * USARTx)

Function description

Get Block Length value in reception.

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_GetBlockLength

LL_USART_EnableIrda

Function name

__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)

Function description

Enable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name

__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)

Function description

Disable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (USART_TypeDef * USARTx)

Function description

Indicate if IrDA mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name

__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)

Function description

Configure IrDA Power Mode (Normal or Low Power)

Parameters

- **USARTx:** USART Instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_IRDA_POWER_LOW

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (USART_TypeDef * USARTx)

Function description

Retrieve IrDA Power Mode configuration (Normal or Low Power)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name

__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (USART_TypeDef * USARTx)

Function description

Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Enable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Disable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (USART_TypeDef * USARTx)

Function description

Indicate if Smartcard NACK transmission is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name

__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)

Function description

Enable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name

__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)

Function description

Disable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (USART_TypeDef * USARTx)

Function description

Indicate if Smartcard mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name

__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)

Function description

Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance
- **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (USART_TypeDef * USARTx)

Function description

Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name

__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (USART_TypeDef * USARTx)

Function description

Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name

__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)

Function description

Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance
- **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (USART_TypeDef * USARTx)

Function description

Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (USART_TypeDef * USARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_EnableSPISlave

Function name

__STATIC_INLINE void LL_USART_EnableSPISlave (USART_TypeDef * USARTx)

Function description

Enable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_EnableSPISlave

LL_USART_DisableSPISlave

Function name

__STATIC_INLINE void LL_USART_DisableSPISlave (USART_TypeDef * USARTx)

Function description

Disable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_DisableSPISlave

LL_USART_IsEnabledSPISlave

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlave (USART_TypeDef * USARTx)

Function description

Indicate if SPI Synchronous Slave mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_IsEnabledSPISlave

LL_USART_EnableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_EnableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Enable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave Selection depends on NSS input pin (The slave is selected when NSS is low and deselected when NSS is high).

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_EnableSPISlaveSelect

LL_USART_DisableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_DisableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Disable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave will be always selected and NSS input pin will be ignored.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_DisableSPISlaveSelect

LL_USART_IsEnabledSPISlaveSelect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Indicate if SPI Slave Selection depends on NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_IsEnabledSPISlaveSelect

LL_USART_SetLINBrkDetectionLen

Function name

__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)

Function description

Set LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance
- **LINBDLength:** This parameter can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (USART_TypeDef * USARTx)

Function description

Return LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name

__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)

Function description

Enable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name

__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)

Function description

Disable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (USART_TypeDef * USARTx)

Function description

Indicate if LIN mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (USART_TypeDef * USARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (USART_TypeDef * USARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_GetDEAssertionTime

LL_USART_EnableDEMode

Function name

__STATIC_INLINE void LL_USART_EnableDEMode (USART_TypeDef * USARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_EnableDEMode

LL_USART_DisableDEMode

Function name

__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_DisableDEMode

LL_USART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (USART_TypeDef * USARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_IsEnabledDEMode

LL_USART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_SetDESignalPolarity

LL_USART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity (USART_TypeDef * USARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_GetDESignalPolarity

LL_USART_ConfigAsyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode
- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name

__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in LIN Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.
- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function
- Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_ConfigLINMode
- CR2 STOP LL_USART_ConfigLINMode
- CR2 LINEN LL_USART_ConfigLINMode
- CR3 IREN LL_USART_ConfigLINMode
- CR3 SCEN LL_USART_ConfigLINMode
- CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name

__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Half Duplex Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name

__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name

__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (USART_TypeDef * USARTx)

Function description

Check if the USART Framing Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (USART_TypeDef * USARTx)

Function description

Check if the USART Noise error detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (USART_TypeDef * USARTx)

Function description

Check if the USART OverRun Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (USART_TypeDef * USARTx)

Function description

Check if the USART IDLE line detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Check if the USART Read Data Register or USART RX FIFO Not Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_USART_IsActiveFlag_RXNE_RXFNE

LL_USART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Data Register Empty or USART TX FIFO Not Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_USART_IsActiveFlag_TXE_TXFNF

LL_USART_IsActiveFlag_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (USART_TypeDef * USARTx)

Function description

Check if the USART CTS interrupt Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTSIF LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS (USART_TypeDef * USARTx)

Function description

Check if the USART CTS Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTS LL_USART_IsActiveFlag_CTS

LL_USART_IsActiveFlag_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO (USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Time Out Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RTOF LL_USART_IsActiveFlag_RTO

LL_USART_IsActiveFlag_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_UDR (USART_TypeDef * USARTx)

Function description

Check if the SPI Slave Underrun error flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR UDR LL_USART_IsActiveFlag_UDR

LL_USART_IsActiveFlag_ABRE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (USART_TypeDef * USARTx)

Function description

Check if the USART Busy Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (USART_TypeDef * USARTx)

Function description

Check if the USART Send Break Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (USART_TypeDef * USARTx)

Function description

Check if the USART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from stop mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (USART_TypeDef * USARTx)

Function description

Check if the USART Receive Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFE (USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFE LL_USART_IsActiveFlag_TXFE

LL_USART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFF (USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFF LL_USART_IsActiveFlag_RXFF

LL_USART_IsActiveFlag_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TCBGT (USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCBGT LL_USART_IsActiveFlag_TCBGT

LL_USART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFT (USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFT LL_USART_IsActiveFlag_TXFT

LL_USART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFT (USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFT LL_USART_IsActiveFlag_RXFT

LL_USART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)

Function description

Clear Parity Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)

Function description

Clear Framing Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)

Function description

Clear Noise Error detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TXFE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TXFE (USART_TypeDef * USARTx)

Function description

Clear TX FIFO Empty Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR TXFECF LL_USART_ClearFlag_TXFE

LL_USART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_TCBGT

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TCBGT (USART_TypeDef * USARTx)

Function description

Clear Smartcard Transmission Complete Before Guard Time Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCBGTCF LL_USART_ClearFlag_TCBGT

LL_USART_ClearFlag_LBD

Function name

__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)

Function description

Clear LIN Break Detection Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name

__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)

Function description

Clear Receiver Time Out Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name

__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)

Function description

Clear End Of Block Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_USART_ClearFlag_UDR (USART_TypeDef * USARTx)

Function description

Clear SPI Slave Underrun Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR UDRCF LL_USART_ClearFlag_UDR

LL_USART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)

Function description

Clear Character Match Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_USART_ClearFlag_CM

LL_USART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR WUCF LL_USART_ClearFlag_WKUP

LL_USART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_EnableIT_RXNE_RXFNE

LL_USART_EnableIT_TC

Function name

__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_EnableIT_TXE_TXFNF

LL_USART_EnableIT_PE

Function name

__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name

__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name

__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name

__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)

Function description

Enable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_EnableIT_EOB

LL_USART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFE (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_EnableIT_TXFE

LL_USART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFF (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_EnableIT_RXFF

LL_USART_EnableIT_LBD

Function name

__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)

Function description

Enable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)

Function description

Enable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)

Function description

Enable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFT (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_EnableIT_TXFT

LL_USART_EnableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Enable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_EnableIT_TCBGT

LL_USART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFT (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_EnableIT_RXFT

LL_USART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_DisableIT_RXNE_RXFNE

LL_USART_DisableIT_TC

Function name

__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_DisableIT_TXE_TXFNF

LL_USART_DisableIT_PE

Function name

__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_CM

Function name

__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_DisableIT_CM

LL_USART_DisableIT_RTO

Function name

__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_DisableIT_RTO

LL_USART_DisableIT_EOB

Function name

__STATIC_INLINE void LL_USART_DisableIT_EOB (USART_TypeDef * USARTx)

Function description

Disable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_DisableIT_EOB

LL_USART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFE (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_DisableIT_TXFE

LL_USART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFF (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_DisableIT_RXFF

LL_USART_DisableIT_LBD

Function name

__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)

Function description

Disable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)

Function description

Disable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)

Function description

Disable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_DisableIT_CTS

LL_USART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_DisableIT_WKUP

LL_USART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFT (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_DisableIT_TXFT

LL_USART_DisableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Disable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_DisableIT_TCBGT

LL_USART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFT (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_DisableIT_RXFT

LL_USART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (USART_TypeDef * USARTx)

Function description

Check if the USART IDLE Interrupt source is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Check if the USART RX Not Empty and USART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_IsEnabledIT_RXNE_RXFNE

LL_USART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Check if the USART TX Empty and USART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_IsEnabledIT_TXE_TXFNF

LL_USART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Timeout Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFE (USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_IsEnabledIT_TXFE

LL_USART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFF (USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_IsEnabledIT_RXFF

LL_USART_IsEnabledIT_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (USART_TypeDef * USARTx)

Function description

Check if the USART Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (USART_TypeDef * USARTx)

Function description

Check if the USART CTS Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFT (USART_TypeDef * USARTx)

Function description

Check if USART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_IsEnabledIT_TXFT

LL_USART_IsEnabledIT_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TCBGT (USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_IsEnabledIT_TCBGT

LL_USART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFT (USART_TypeDef * USARTx)

Function description

Check if USART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_IsEnabledIT_RXFT

LL_USART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (USART_TypeDef * USARTx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **USARTx:** USART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_USART_DMA_REG_DATA_TRANSMIT
  - LL_USART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_DMA_GetRegAddr
-
- TDR TDR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name

__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name

__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)

Function description

Request an Automatic Baud Rate measurement on next received data frame.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name

__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)

Function description

Request Break sending.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)

Function description

Put USART in mute mode and set the RWU flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Transmit data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_DeInit

Function name

ErrorStatus LL_USART_DeInit (USART_TypeDef * USARTx)

Function description

De-initialize USART registers (Registers restored to their default values).

Parameters

- **USARTx:** USART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are de-initialized
  - ERROR: USART registers are not de-initialized

LL_USART_Init

Function name

ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, LL_USART_InitTypeDef * USART_InitStruct)

Function description

Initialize USART registers according to the specified parameters in USART_InitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are initialized according to USART_InitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name

void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)

Function description

Set each LL_USART_InitTypeDef field to default value.

Parameters

- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_USART_ClockInit

Function name

ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name

void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Set each field of a LL_USART_ClockInitTypeDef type structure to default value.

Parameters

- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

92.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

92.3.1 USART

USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_USART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT

Measurement of the start bit is used to detect the baud rate

LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE

Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx

LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME

0x7F frame detection

LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

0x55 frame detection

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_USART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_USART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF

Parity error flag

LL_USART_ICR_FECF

Framing error flag

LL_USART_ICR_NECF

Noise error detected flag

LL_USART_ICR_ORECF

Overrun error flag

LL_USART_ICR_IDLECF

Idle line detected flag

LL_USART_ICR_TXFECF

TX FIFO Empty Clear flag

LL_USART_ICR_TCCF

Transmission complete flag

LL_USART_ICR_TCBGTCF

Transmission completed before guard time flag

LL_USART_ICR_LBDCF

LIN break detection flag

LL_USART_ICR_CTSCF

CTS flag

LL_USART_ICR_RTOCF

Receiver timeout flag

LL_USART_ICR_EOBCF

End of block flag

LL_USART_ICR_UDRCF

SPI Slave Underrun Clear flag

LL_USART_ICR_CMCF

Character match flag

LL_USART_ICR_WUCF

Wakeup from Stop mode flag

Clock Signal

LL_USART_CLOCK_DISABLE

Clock signal not provided

LL_USART_CLOCK_ENABLE

Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_USART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_USART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH

DE signal is active high

LL_USART_DE_POLARITY_LOW

DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_USART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_USART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_USART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_USART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_USART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_USART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_USART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_USART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_USART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_USART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_USART_ISR_PE

Parity error flag

LL_USART_ISR_FE

Framing error flag

LL_USART_ISR_NE

Noise detected flag

LL_USART_ISR_ORE

Overrun error flag

LL_USART_ISR_IDLE

Idle line detected flag

LL_USART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_USART_ISR_TC

Transmission complete flag

LL_USART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_USART_ISR_LBDF

LIN break detection flag

LL_USART_ISR_CTSIF

CTS interrupt flag

LL_USART_ISR_CTS

CTS flag

LL_USART_ISR_RTOF

Receiver timeout flag

LL_USART_ISR_EOBF

End of block flag

LL_USART_ISR_UDR

SPI Slave underrun error flag

LL_USART_ISR_ABRE

Auto baud rate error flag

LL_USART_ISR_ABRF

Auto baud rate flag

LL_USART_ISR_BUSY

Busy flag

LL_USART_ISR_CMF

Character match flag

LL_USART_ISR_SBKF

Send break flag

LL_USART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_USART_ISR_WUF

Wakeup from Stop mode flag

LL_USART_ISR_TEACK

Transmit enable acknowledge flag

LL_USART_ISR_REACK

Receive enable acknowledge flag

LL_USART_ISR_TXFE

TX FIFO empty flag

LL_USART_ISR_RXFF

RX FIFO full flag

LL_USART_ISR_TCBGT

Transmission complete before guard time completion flag

LL_USART_ISR_RXFT

RX FIFO threshold flag

LL_USART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_USART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_USART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_USART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_USART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL

IrDA normal power mode

LL_USART_IRDA_POWER_LOW

IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE

IDLE interrupt enable

LL_USART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_USART_CR1_TCIE

Transmission complete interrupt enable

LL_USART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_USART_CR1_PEIE

Parity error

LL_USART_CR1_CMIE

Character match interrupt enable

LL_USART_CR1_RTOIE

Receiver timeout interrupt enable

LL_USART_CR1_EOBIE

End of Block interrupt enable

LL_USART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_USART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_USART_CR2_LBDIE

LIN break detection interrupt enable

LL_USART_CR3_EIE

Error interrupt enable

LL_USART_CR3_CTSIE

CTS interrupt enable

LL_USART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_USART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_USART_CR3_TCBGTIE

Transmission complete before guard time interrupt enable

LL_USART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT

The clock pulse of the last data bit is not output to the SCLK pin

LL_USART_LASTCLKPULSE_OUTPUT

The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B

10-bit break detection method selected

LL_USART_LINBREAK_DETECT_11B

11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16

Oversampling by 16

LL_USART_OVERSAMPLING_8

Oversampling by 8

Parity Control

LL_USART_PARITY_NONE

Parity control disabled

LL_USART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_USART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE

The first clock transition is the first data capture edge

LL_USART_PHASE_2EDGE

The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW

Steady low value on SCLK pin outside transmission window

LL_USART_POLARITY_HIGH

Steady high value on SCLK pin outside transmission window

Clock Source Prescaler

LL_USART_PRESCALER_DIV1

Input clock not devided

LL_USART_PRESCALER_DIV2

Input clock devided by 2

LL_USART_PRESCALER_DIV4

Input clock devided by 4

LL_USART_PRESCALER_DIV6

Input clock devided by 6

LL_USART_PRESCALER_DIV8

Input clock devided by 8

LL_USART_PRESCALER_DIV10

Input clock devided by 10

LL_USART_PRESCALER_DIV12

Input clock devided by 12

LL_USART_PRESCALER_DIV16

Input clock devided by 16

LL_USART_PRESCALER_DIV32

Input clock devided by 32

LL_USART_PRESCALER_DIV64

Input clock devided by 64

LL_USART_PRESCALER_DIV128

Input clock devided by 128

LL_USART_PRESCALER_DIV256

Input clock devided by 256

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_USART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5

0.5 stop bit

LL_USART_STOPBITS_1

1 stop bit

LL_USART_STOPBITS_1_5

1.5 stop bits

LL_USART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_USART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_USART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE

USART wake up from Mute mode on Idle Line

LL_USART_WAKEUP_ADDRESSMARK

USART wake up from Mute mode on Address Mark

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_USART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_USART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TXE

IT_Management

LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TXE

LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TXE

LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TXE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

93 LL UTILS Generic Driver

93.1 UTILS Firmware driver registers structures

93.1.1 LL_UTILS_PLLInitTypeDef

LL_UTILS_PLLInitTypeDef is defined in the stm32g4xx_ll_utils.h

Data Fields

- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLR***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLM***

  Division factor for PLL VCO input clock. This parameter can be a value of ***RCC_LL_EC_PLLM_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLN***

  Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 8 and Max_Data = 86This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLR***

  Division for the main system clock. This parameter can be a value of ***RCC_LL_EC_PLLR_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL_ConfigDomain_SYS()**.

93.1.2 LL_UTILS_ClkInitTypeDef

LL_UTILS_ClkInitTypeDef is defined in the stm32g4xx_ll_utils.h

Data Fields

- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***

  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***

  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***

  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.

93.2 UTILS Firmware driver API description

The following section lists the various functions of the UTILS library.

93.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 170000000 Hz for STM32G4xx.

This section contains the following APIs:

- *LL_SetSystemCoreClock*
- *LL_SetFlashLatency*
- *LL_PLL_ConfigSystemClock_HSI*
- *LL_PLL_ConfigSystemClock_HSE*

93.2.2 Detailed description of functions

LL_GetUID_Word0

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)

Function description

Get Word0 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[31:0]:** X and Y coordinates on the wafer expressed in BCD format

LL_GetUID_Word1

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)

Function description

Get Word1 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[63:32]:** Wafer number (UID[39:32]) & LOT_NUM[23:0] (UID[63:40])

LL_GetUID_Word2

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)

Function description

Get Word2 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[95:64]:** Lot number (ASCII encoded) - LOT_NUM[55:24]

LL_GetFlashSize

Function name

__STATIC_INLINE uint32_t LL_GetFlashSize (void)

Function description

Get Flash memory size.

Return values

- **FLASH_SIZE[15:0]:** Flash memory size

Notes

- This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_GetPackageType

Function name

__STATIC_INLINE uint32_t LL_GetPackageType (void)

Function description

Get Package type.

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_PACKAGETYPE_LQFP64
  - LL_UTILS_PACKAGETYPE_LQFP100
  - LL_UTILS_PACKAGETYPE_WLCSP81
  - LL_UTILS_PACKAGETYPE_LQFP128
  - LL_UTILS_PACKAGETYPE_UFQFPN32
  - LL_UTILS_PACKAGETYPE_LQFP32
  - LL_UTILS_PACKAGETYPE_UFQFPN48
  - LL_UTILS_PACKAGETYPE_LQFP48
  - LL_UTILS_PACKAGETYPE_WLCSP49
  - LL_UTILS_PACKAGETYPE_UFBGA64
  - LL_UTILS_PACKAGETYPE_UFBGA100
  - LL_UTILS_PACKAGETYPE_LQFP48_EBIKE

LL_InitTick

Function name

__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)

Function description

This function configures the Cortex-M SysTick source of the time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
- **Ticks:** Number of ticks

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name

void LL_Init1msTick (uint32_t HCLKFrequency)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.
- HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name

void LL_mDelay (uint32_t Delay)

Function description

This function provides accurate delay (in milliseconds) based on SysTick counter flag.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service.
- To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name

void LL_SetSystemCoreClock (uint32_t HCLKFrequency)

Function description

This function sets directly SystemCoreClock CMSIS variable.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)

Return values

- **None:**

Notes

- Variable can be calculated also through SystemCoreClockUpdate function.

LL_SetFlashLatency

Function name

ErrorStatus LL_SetFlashLatency (uint32_t HCLKFrequency)

Function description

Update number of Flash wait states in line with new frequency and current voltage range.

Parameters

- **HCLKFrequency:** HCLK frequency

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Latency has been modified
  - ERROR: Latency cannot be modified

LL_PLL_ConfigSystemClock_HSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock at maximum frequency with HSI as clock source of the PLL.

Parameters

- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: PLL output frequency = (((HSI frequency / PLLM) * PLLN) / PLLR)PLLM: ensure that the VCO input frequency ranges from 2.66 to 8 MHz (PLLVCO_input = HSI frequency / PLLM)PLLN: ensure that the VCO output frequency is between 64 and 344 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLR: ensure that max frequency at 170000000 Hz is reach (PLLVCO_output / PLLR)

LL_PLL_ConfigSystemClock_HSE

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock with HSE as clock source of the PLL.

Parameters

- **HSEFrequency:** Value between Min_Data = 4000000 and Max_Data = 48000000
- **HSEBypass:** This parameter can be one of the following values:
  - LL_UTILS_HSEBYPASS_ON
  - LL_UTILS_HSEBYPASS_OFF
- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: PLL output frequency = (((HSE frequency / PLLM) * PLLN) / PLLR)PLLM: ensure that the VCO input frequency ranges from 2.66 to 8 MHz (PLLVCO_input = HSE frequency / PLLM)PLLN: ensure that the VCO output frequency is between 64 and 344 MHz (PLLVCO_output = PLLVCO_input * PLLN)PLLR: ensure that max frequency at 170000000 Hz is reached (PLLVCO_output / PLLR)

93.3 UTILS Firmware driver defines

The following section lists the various define and macros of the module.

93.3.1 UTILS

UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF

HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON

HSE Bypass is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_LQFP64

LQFP64 package type

LL_UTILS_PACKAGETYPE_LQFP100

LQFP100 package type

LL_UTILS_PACKAGETYPE_WLCSP81

WLCSP81 package type

LL_UTILS_PACKAGETYPE_LQFP128

LQFP128 package type

LL_UTILS_PACKAGETYPE_UFQFPN32

UFQFPN32 package type

LL_UTILS_PACKAGETYPE_LQFP32

LQFP32 package type

LL_UTILS_PACKAGETYPE_UFQFPN48

UFQFPN48 package type

LL_UTILS_PACKAGETYPE_LQFP48

LQFP48 package type

LL_UTILS_PACKAGETYPE_WLCSP49

WLCSP49 package type

LL_UTILS_PACKAGETYPE_UFBGA64

UFBGA64 package type

LL_UTILS_PACKAGETYPE_UFBGA100

UFBGA100 package type

LL_UTILS_PACKAGETYPE_LQFP48_EBIKE

LQFP48 EBIKE package type

94 LL WWDG Generic Driver

94.1 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

94.1.1 Detailed description of functions

LL_WWDG_Enable

Function name

__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)

Function description

Enable Window Watchdog.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)

Function description

Checks if Window Watchdog is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name

__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)

Function description

Set the Watchdog counter value to provided value (7-bits T[6:0])

Parameters

- **WWDGx:** WWDG Instance
- **Counter:** 0..0x7F (7 bit counter value)

Return values

- **None:**

Notes

- When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6 becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)

Reference Manual to LL API cross reference:

- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetCounter (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Counter Value (7 bits counter value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Counter value

Reference Manual to LL API cross reference:

- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)

Function description

Set the time base of the prescaler (WDGTB).

Parameters

- **WWDGx:** WWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Return values

- **None:**

Notes

- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Prescaler Value.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name

__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description

Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters

- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values

- **None:**

Notes

- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Window Value (7 bits value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Enable the Early Wakeup Interrupt.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Check if Early Wakeup Interrupt is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

94.2 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

94.2.1 WWDG

WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

LL_WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

LL_WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

LL_WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

LL_WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

LL_WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

LL_WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

LL_WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

Common Write and read registers macros

LL_WWDG_WriteReg

Description:

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_WWDG_ReadReg

Description:

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

95 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 Series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, …). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32G4 devices. To ensure compatibility between all devices and portability with others Series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *section* ***Devices supported by the HAL drivers***.

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32g4xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32g4xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32g4xx_hal.h file has to be included.

What is the difference between xx_hal_ppp.c/.h and xx_hal_ppp_ *ex* .c/.h?

The HAL driver architecture supports common features across STM32 Series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32g4xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines

- The extension APIs (stm32g4xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32g4xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32g4xx_hal_msp.c. A template is provided in the HAL driver folders (stm32g4xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute *__weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() instm32g4xx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32g4xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32g4xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32g4xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

Revision history

Table 25. **Document revision history**

Date	Version	Changes
14-May-2019	1	Initial release.
03-Aug-2020	2	Updated Section 2 Acronyms and definitions. Added STM32G491xx and STM32G4A1xx part numbers in Table 5. List of devices supported by HAL drivers and Section 3.5.1 HAL API naming rules. Updated HAL and LL drivers.

Contents

List of tables

List of figures