

---

## Description of STM32F4 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 **STM32CubeF4** for STM32F4 Series)
  - 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, USB, TCP/IP and Graphics.
  - 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 CodeSona<sup>®</sup> static analysis tool. It is fully documented.

It is compliant with MISRA C<sup>®</sup>:2004 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 STM32CubeF4 MCU Package runs on STM32F4 32-bit microcontrollers based on the Arm<sup>®</sup> Cortex<sup>®</sup>-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
ESE	Security enable Flash user option bit
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 security controller
GTZC-MPCBB	GTZC block-based memory protection controller
GTZC-MPCWM	GTZC watermark memory protection controller
GTZC-TZIC	Security illegal access controller
GTZC-TZSC	Security access 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 controller
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
SMARTCARD	Smartcard IC

Acronym	Definition
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
UART	Universal asynchronous receiver/transmitter
UCPD	USB Type-C <sup>®</sup> 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/DelInit 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
<i>stm32f4xx_hal_ppp.c</i>	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. <i>Example: stm32f4xx_hal_adc.c, stm32f4xx_hal_irda.c.</i>
<i>stm32f4xx_hal_ppp.h</i>	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. <i>Example: stm32f4xx_hal_adc.h, stm32f4xx_hal_irda.h.</i>
<i>stm32f4xx_hal_ppp_ex.c</i>	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. <i>Example: stm32f4xx_hal_adc_ex.c, stm32f4xx_hal_flash_ex.c.</i>
<i>stm32f4xx_hal_ppp_ex.h</i>	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 <i>Example: stm32f4xx_hal_adc_ex.h, stm32f4xx_hal_flash_ex.h.</i>
<i>stm32f4xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
<i>stm32f4xx_hal.h</i>	<i>stm32f4xx_hal.c</i> header file
<i>stm32f4xx_hal_msp_template.c</i>	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.
<i>stm32f4xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application
<i>stm32f4xx_hal_def.h</i>	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
<i>system_stm32f4xx.c</i>	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 and configuring the FSMC/FMC (when available) to use the external SRAM or SDRAM mounted on the evaluation board as data memory.
<i>startup_stm32f4xx.s</i>	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.
<i>stm32f4xx_flash.icf</i> (optional)	Linker file for EWARM toolchain allowing mainly adapting the stack/heap size to fit the application requirements.
<i>stm32f4xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.



File	Description
<i>stm32f4xx_hal_conf.h</i>	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.
<i>stm32f4xx_it.c/h</i>	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 <i>stm32f4xx_hal.c</i> ) 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.
<i>main.c/h</i>	This file contains the main program routine, mainly: <ul style="list-style-type: none"> <li>• Call to HAL_Init()</li> <li>• assert_failed() implementation</li> <li>• system clock configuration</li> <li>• peripheral HAL initialization and user application code.</li> </ul>

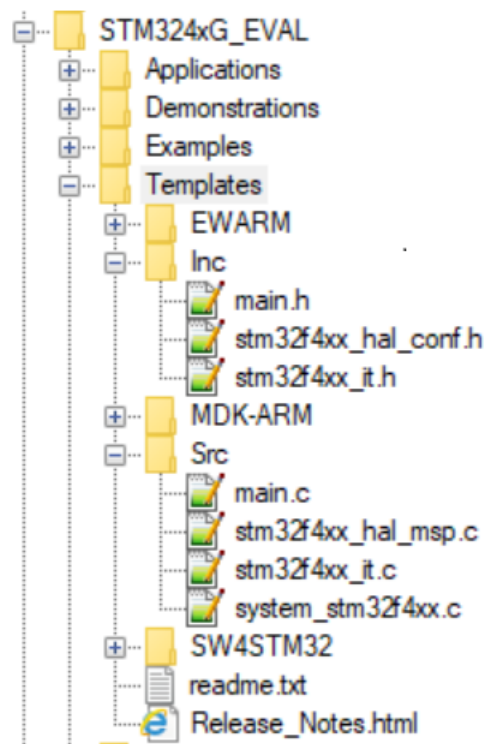
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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
  - SysTick ISR implemented for HAL\_Delay()
  - System clock configured with the maximum device frequency.

*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 enables 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 baudrate.*/
  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 fPCLK/8).*/
}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 the following 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_InjectedStop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEX_InjectedStop_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEX_InjectedStart(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEX_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

- **Device part number specific APIs:** These APIs are implemented in the extension file and delimited by specific define statements relative to a given part number.

```
#if defined(STM32F427xx) || defined(STM32F437xx) || defined(STM32F429xx)
|| defined(STM32F439xx) HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);
#endif /* STM32F427xx || STM32F437xx || STM32F429xx || STM32F439xx || */
```

**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
<b>Common APIs</b>	X	X <sup>(1)</sup>
<b>Family specific APIs</b>	-	X
<b>Device specific APIs</b>	-	X

1. *In some cases, the implementation for a specific device part number may change. In this case the generic API is declared as weak function in the extension file. The API is implemented again to overwrite the default function.*

**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	STM32F405xx	STM32F415xx	STM32F407xx	STM32F417xx	STM32F427xx	STM32F437xx	STM32F429xx	STM32F439xx	STM32F401xC	STM32F401xE	STM32F446xx	STM32F469xx	STM32F479xx	STM32F410xx	STM32F412xx
stm32f4xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_can.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_cec.c	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No
stm32f4xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_cryp.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No	No	No	Yes	No	No
stm32f4xx_hal_cryp_ex.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No	No	No	Yes	No	No
stm32f4xx_hal_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No
stm32f4xx_hal_dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No
stm32f4xx_hal_dcmi.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_hal_dcmi_ex.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_hal_ufs.c	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes
stm32f4xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_dma2d.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_dsi.c	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_eth.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_flash_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_flash_ramfunc.c	No	No	No	No	No	No	No	No	No	No	Yes	No	No	Yes	Yes
stm32f4xx_hal_fmapi2c.c	No	No	No	No	No	No	No	No	No	No	Yes	No	No	Yes	Yes
stm32f4xx_hal_fmapi2c_ex.c	No	No	No	No	No	No	No	No	No	No	Yes	No	No	Yes	Yes
stm32f4xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_hash.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No	No	No	Yes	No	No
stm32f4xx_hal_hash_ex.c	No	No	No	No	No	Yes	No	Yes	No	No	No	No	Yes	No	No
stm32f4xx_hal_hcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_i2c_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_i2s.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_i2s_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32F405xx	STM32F415xx	STM32F407xx	STM32F417xx	STM32F427xx	STM32F437xx	STM32F429xx	STM32F439xx	STM32F401xC	STM32F401xE	STM32F446xx	STM32F469xx	STM32F479xx	STM32F410xx	STM32F412xx
stm32f4xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_lptim.c	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No
stm32f4xx_hal_itdc.c	No	No	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_itdc_ex.c	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_msp_template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32f4xx_hal_nand.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_hal_nor.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_pccard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No
stm32f4xx_hal_pcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_pcd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_pwr_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_qspi.c	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
stm32f4xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_sai.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_hal_sai_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_hal_sd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_sdram.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_spdifrx.c	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No
stm32f4xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_sram.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes
stm32f4xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_ll_fmc.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
stm32f4xx_ll_fsmc.c	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	Yes
stm32f4xx_ll_sdmmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
stm32f4xx_ll_usb.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	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	<i>stm32f4xx_hal_ppp (c/h)</i>	<i>stm32f4xx_hal_ppp_ex (c/h)</i>	<i>stm32f4xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	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 STM32F4 reference manuals.
- Peripheral registers are declared in the `PPP_TypeDef` structure (for example `ADC_TypeDef`) in the `stm32f4xxx.h` header file:  
`stm32f4xxx.h` corresponds to `stm32f401xc.h`, `stm32f401xe.h`, `stm32f405xx.h`, `stm32f415xx.h`, `stm32f407xx.h`, `stm32f417xx.h`, `stm32f427xx.h`, `stm32f437xx.h`, `stm32f429xx.h`, `stm32f439xx.h`, `stm32f446xx.h`, `stm32f469xx.h`, `stm32f479xx.h`, `stm32f410cx.h`, `stm32f410tx.h`, `stm32f410rx.h`, `stm32f412cx.h`, `stm32f412rx.h`, `stm32f412vx.h` or `stm32f412zx.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.  
 Example: `HAL_ADCEx_InjectedStart()` refers to the injection mode.

### 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
<code>__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Enables a specific peripheral interrupt
<code>__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Disables a specific peripheral interrupt
<code>__HAL_PPP_GET_IT(__HANDLE__, __INTERRUPT__)</code>	Gets a specific peripheral interrupt status
<code>__HAL_PPP_CLEAR_IT(__HANDLE__, __INTERRUPT__)</code>	Clears a specific peripheral interrupt status
<code>__HAL_PPP_GET_FLAG(__HANDLE__, __FLAG__)</code>	Gets a specific peripheral flag status
<code>__HAL_PPP_CLEAR_FLAG(__HANDLE__, __FLAG__)</code>	Clears a specific peripheral flag status
<code>__HAL_PPP_ENABLE(__HANDLE__)</code>	Enables a peripheral
<code>__HAL_PPP_DISABLE(__HANDLE__)</code>	Disables a peripheral
<code>__HAL_PPP_XXXX(__HANDLE__, __PARAM__)</code>	Specific PPP HAL driver macro
<code>__HAL_PPP_GET_IT_SOURCE(__HANDLE__, __INTERRUPT__)</code>	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 `stm32f4xx_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 stm32f4xx\_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
<i>Initialization group</i>	<i>HAL_ADC_Init()</i>	This function initializes the peripheral and configures the low-level resources (clocks, GPIO, AF..)
	<i>HAL_ADC_DeInit()</i>	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
<i>IO operation group</i>	<i>HAL_ADC_Start ()</i>	This function starts ADC conversions when the polling method is used
	<i>HAL_ADC_Stop ()</i>	This function stops ADC conversions when the polling method is used
	<i>HAL_ADC_PollForConversion()</i>	This function allows waiting for the end of conversions when the polling method is used. In this case, a timeout value is specified by the user according to the application.
	<i>HAL_ADC_Start_IT()</i>	This function starts ADC conversions when the interrupt method is used
	<i>HAL_ADC_Stop_IT()</i>	This function stops ADC conversions when the interrupt method is used
	<i>HAL_ADC_IRQHandler()</i>	This function handles ADC interrupt requests
	<i>HAL_ADC_ConvCpltCallback()</i>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
<i>Control group</i>	<i>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
	<i>HAL_ADC_ConfigChannel()</i>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
<i>State and Errors group</i>	<i>HAL_ADC_AnalogWDGConfig</i>	This function configures the analog watchdog for the selected ADC
	<i>HAL_ADC_GetState()</i>	This function allows getting in run time the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function 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, *stm32f4xx\_hal\_ppp\_ex.c*, that includes all the specific functions and define statements (*stm32f4xx\_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
<i>HAL_ADCEx_InjectedStart()</i>	This function starts injected channel ADC conversions when the polling method is used
<i>HAL_ADCEx_InjectedStop()</i>	This function stops injected channel ADC conversions when the polling method is used

Function Group	Common API Name
<i>HAL_ADCEx_InjectedStart_IT()</i>	This function starts injected channel ADC conversions when the interrupt method is used
<i>HAL_ADCEx_InjectedStop_IT()</i>	This function stops injected channel ADC conversions when the interrupt method is used
<i>HAL_ADCEx_InjectedConfigChannel()</i>	This function configures the selected ADC Injected channel (corresponding rank in the sequencer and sample time)

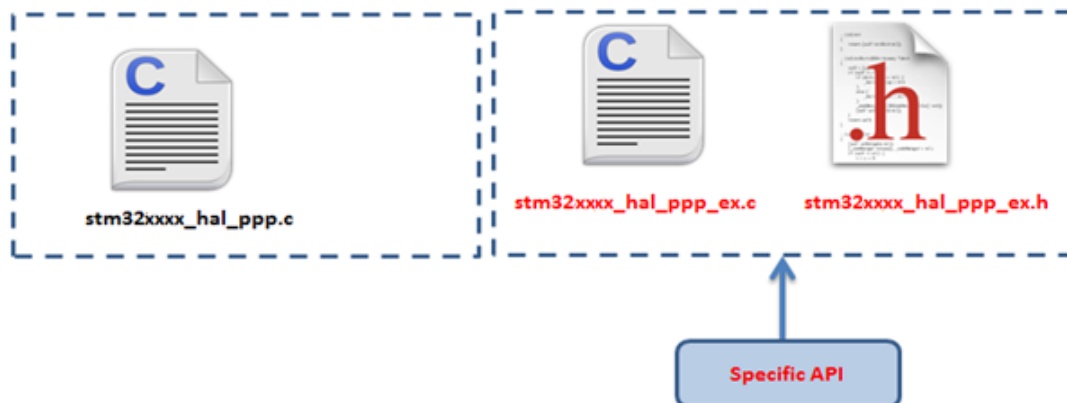
### 3.7.2 HAL extension model cases

The specific peripheral 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 `stm32f4xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEX_Function()`.

Figure 2. Adding device-specific functions



Example: `stm32f4xx_hal_flash_ex.c/h`

```
#if defined(STM32F427xx) || defined(STM32F437xx) || defined(STM32F429xx) ||
defined(STM32F439xx)
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);
#endif /* STM32F427xx || STM32F437xx || STM32F429xx || STM32F439xx */
```

#### 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



Example: stm32f4xx\_hal\_adc\_ex.c/h

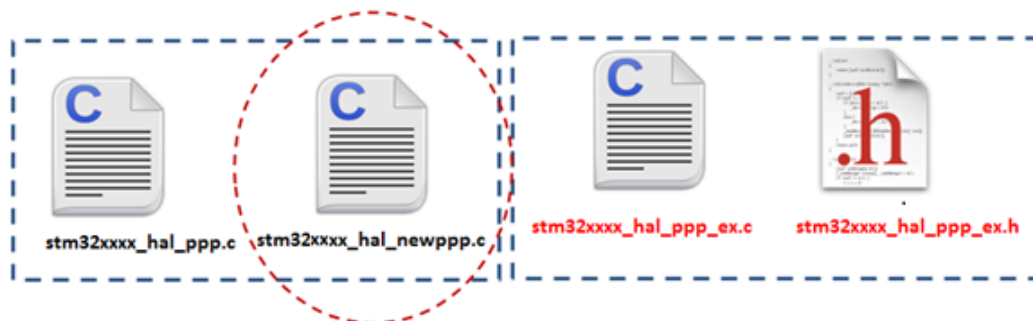
```
HAL_StatusTypeDef HAL_ADCEx_InjectedStop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

### 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 stm32f4xx\_hal\_newppp.c. However the inclusion of this file is selected in the stm32f4xx\_hal\_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. Adding new peripherals

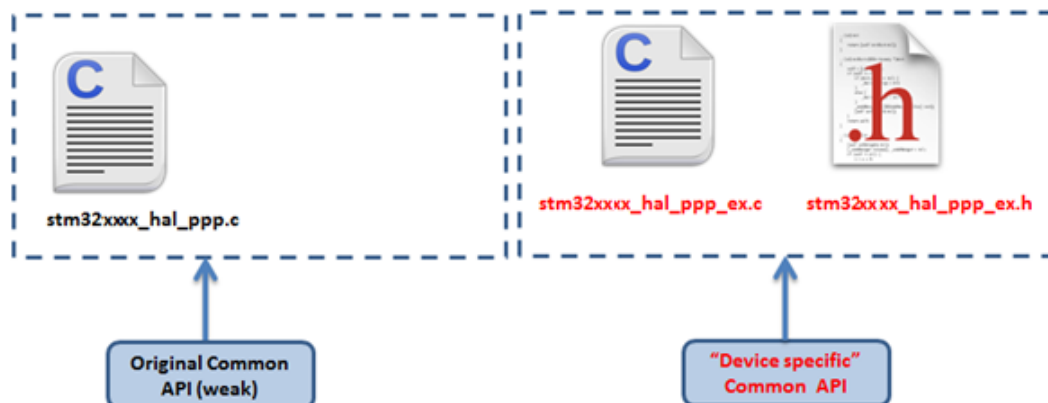


Example: stm32f4xx\_hal\_adc.c/h

### Updating existing common APIs

In this case, the routines are defined with the same names in the stm32f4xx\_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



### Updating existing data structures

The data structure for a specific device part number (for example PPP\_InitTypeDef) can have different fields. In this case, the data structure is defined in the extension header file and delimited by the specific part number define statement.

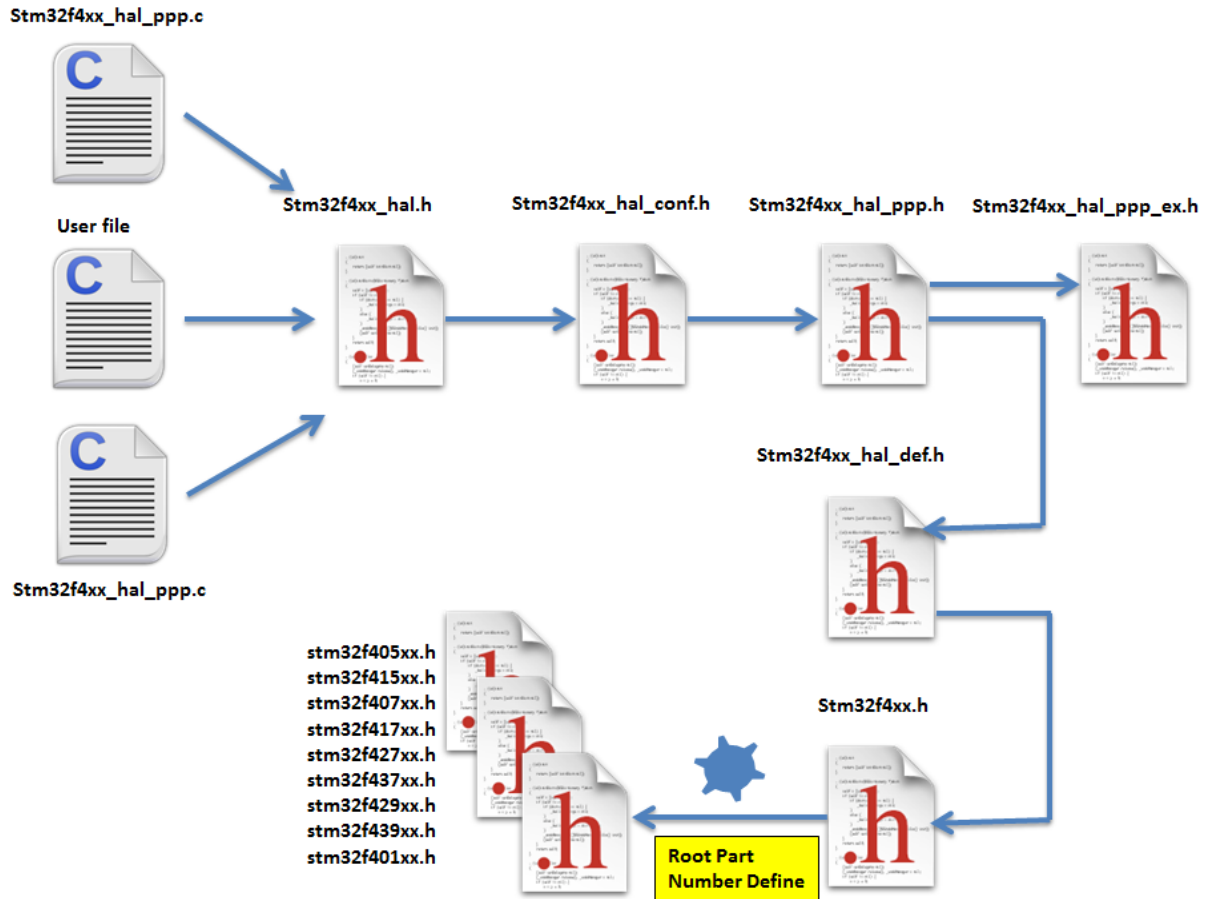
Example:

```
#if defined (STM32F401xx)
typedef struct
{
  (...)
}PPP_InitTypeDef;
#endif /* STM32F401xx */
```

### 3.8 File inclusion model

The header of the common HAL driver file (stm32f4xx\_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 stm32f4xx_hal_conf.h
 * @author MCD Application Team
 * @version VX.Y.Z * @date dd-mm-yyyy
 * @brief This file contains the modules to be used
 *****/
(...)
#define USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
#define USE_HAL_RCC_MODULE
(...)

```

### 3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32f4xx\_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 *stm32f4xx\_hal\_def.h* file calls the *stm32f4xx.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, *stm32f4xx\_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
<b>HSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 (Hz)
<b>HSE_STARTUP_TIMEOUT</b>	Timeout for HSE start-up, expressed in ms	5000
<b>HSI_VALUE</b>	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)



Configuration item	Description	Default Value
<b>EXTERNAL_CLOCK_VALUE</b>	This value is used by the I2S/SAI HAL module to compute the I2S/SAI clock source frequency. This source is inserted directly through I2S_CKIN pad.	12288000 (Hz)
<b>VDD_VALUE</b>	VDD value	3300 (mV)
<b>USE_RTOS</b>	Enables the use of RTOS	FALSE (for future use)
<b>PREFETCH_ENABLE</b>	Enables prefetch feature	TRUE
<b>INSTRUCTION_CACHE_ENABLE</b>	Enables instruction cache	TRUE
<b>DATA_CACHE_ENABLE</b>	Enables data cache	TRUE
<b>USE_HAL_PPP_MODULE</b>	Enables module to be used in the HAL driver	
<b>MAC_ADDRx</b>	Ethernet peripheral configuration : MAC address	
<b>ETH_RX_BUF_SIZE</b>	Ethernet buffer size for data reception	ETH_MAX_PACKET_SIZE
<b>ETH_TX_BUF_SIZE</b>	Ethernet buffer size for transmit	ETH_MAX_PACKET_SIZE
<b>ETH_RXBUFNB</b>	The number of Rx buffers of size ETH_RX_BUF_SIZE	4
<b>ETH_TXBUFNB</b>	The number of Tx buffers of size ETH_RX_BUF_SIZE	4
<b>DP83848_PHY_ADDRESS</b>	DB83848 Ethernet PHY Address	0x01
<b>PHY_RESET_DELAY</b>	PHY Reset delay these values are based on a 1 ms SysTick interrupt	0x000000FF
<b>PHY_CONFIG_DELAY</b>	PHY Configuration delay	0x00000FFF
<b>PHY_BCR PHY_BSR</b>	Common PHY Registers	
<b>PHY_SR PHY_MICR PHY_MISR</b>	Extended PHY registers	

*Note:* The `stm32f4xx_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 `stm32f4xx_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 section 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 Clocks

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 AHB, APB1 and APB2 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, SDIO, I2S, Audio, PLL). In this case, the clock configuration is performed by an extended API defined in `stm32f4xx_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 retrieving various clock configurations (such as system clock, HCLK, PCLK1 or PCLK2)
- MCO and CSS configuration functions

A set of macros are defined in `stm32f4xx_hal_rcc.h` and `stm32f4xx_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.

### 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 `stm32f4xx_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: <code>GPIO_PIN_x</code> or <code>GPIO_PIN_All</code> , where <code>x[0..15]</code>
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: <ul style="list-style-type: none"> <li>• <u>GPIO mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_INPUT</code> : Input floating</li> <li>– <code>GPIO_MODE_OUTPUT_PP</code> : Output push-pull</li> <li>– <code>GPIO_MODE_OUTPUT_OD</code> : Output open drain</li> <li>– <code>GPIO_MODE_AF_PP</code> : Alternate function push-pull</li> <li>– <code>GPIO_MODE_AF_OD</code> : Alternate function open drain</li> <li>– <code>GPIO_MODE_ANALOG</code> : Analog mode</li> </ul> </li> <li>• <u>External Interrupt mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_IT_RISING</code> : Rising edge trigger detection</li> <li>– <code>GPIO_MODE_IT_FALLING</code> : Falling edge trigger detection</li> <li>– <code>GPIO_MODE_IT_RISING_FALLING</code> : Rising/Falling edge trigger detection</li> </ul> </li> <li>• <u>External Event mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_EVT_RISING</code> : Rising edge trigger detection</li> <li>– <code>GPIO_MODE_EVT_FALLING</code> : Falling edge trigger detection</li> <li>– <code>GPIO_MODE_EVT_RISING_FALLING</code> : Rising/Falling edge trigger detection</li> </ul> </li> </ul>
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: <code>GPIO_NOPULL</code> <code>GPIO_PULLUP</code> <code>GPIO_PULLDOWN</code>
Speed	Specifies the speed for the selected pins Possible values are: <code>GPIO_SPEED_LOW</code> <code>GPIO_SPEED_MEDIUM</code> <code>GPIO_SPEED_FAST</code>

Structure field	Description
	GPIO_SPEED_HIGH
Alternate	Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PPP, where AFx: is the alternate function index and PPP: is the peripheral instance Example: use GPIO_AF1_TIM2 to connect TIM2 I/Os on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines. <i>Note:</i> Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.

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_FAST;
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);
```

- Configuring USART3 Tx (PC10, mapped on AF7) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
GPIO_InitStruct.Alternate = GPIO_AF7_USART3;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

### 3.11.3 Cortex® NVIC and SysTick timer

The Cortex® HAL driver, stm32f4xx\_hal\_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL\_NVIC\_SetPriorityGrouping()
  - HAL\_NVIC\_SetPriority()
  - HAL\_NVIC\_EnableIRQ()/HAL\_NVIC\_DisableIRQ()
  - HAL\_NVIC\_SystemReset()
  - HAL\_NVIC\_GetPendingIRQ() / HAL\_NVIC\_SetPendingIRQ () / HAL\_NVIC\_ClearPendingIRQ()
  - HAL\_SYSTICK\_Config()
  - HAL\_SYSTICK\_CLKSourceConfig()

### 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\_PWR\_PVD\_IRQHandler()
  - HAL\_PWR\_PVDCallback()

- Wakeup pin configuration
  - HAL\_PWR\_EnableWakeUpPin() / HAL\_PWR\_DisableWakeUpPin()
- Low-power mode entry
  - HAL\_PWR\_EnterSLEEPMode()
  - HAL\_PWR\_EnterSTOPMode()
  - HAL\_PWR\_EnterSTANDBYMode()

Depending on the STM32 Series, extension functions are available in stm32f4xx\_hal\_pwr\_ex. Here are a few examples (the list is not exhaustive):

- Backup domain registers enable/disable
  - HAL\_PWREx\_EnableBkUpReg() / HAL\_PWREx\_DisableBkUpReg()
- Flash overdrive control and Flash power-down (available only on STM32F429/F439xx)
  - HAL\_PWREx\_ActivateOverDrive()
  - HAL\_PWREx\_EnableFlashPowerDown().

### 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, COMP, and USB 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
PPP_EXTI_LINE_FUNCTION	Defines the EXTI line connected to the internal peripheral. Example: <pre>#define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*1&lt;External interrupt line 16 Connected to the PVD EXTI Line */</pre>
__HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)	Enables a given EXTI line Example: <pre>__HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)</pre>
__HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)	Disables a given EXTI line. Example: <pre>__HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)</pre>
__HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)	Gets a given EXTI line interrupt flag pending bit status. Example: <pre>__HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)</pre>
__HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)	Clears a given EXTI line interrupt flag pending bit. Example; <pre>__HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)</pre>
__HAL_PPP_EXTI_GENERATE_SWIT (__EXTI_LINE__)	Generates a software interrupt for a given EXTI line. Example: <pre>__HAL_PVD_EXTI_GENERATE_SWIT (PWR_EXTI_LINE_PVD)</pre>

If the EXTI interrupt mode is selected, the user application must call HAL\_PPP\_FUNCTION\_IRQHandler() (for example HAL\_PWR\_PVD\_IRQHandler()), from stm32f4xx\_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 or peripheral flow control mode
- Channel priority level
- Source and destination Increment mode
- FIFO mode and its threshold (if needed)
- Burst mode for source and/or destination (if needed).

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 configured. 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_FS`: returns the current DMA Stream FIFO filled level.
- `__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 must be done in the HAL\_PPP\_MspInit() callback. In addition, the user application must associate the DMA handle to the PPP handle (refer to section "HAL IO operation functions").

*Note:* DMA double-buffering feature is handled as an extension API.

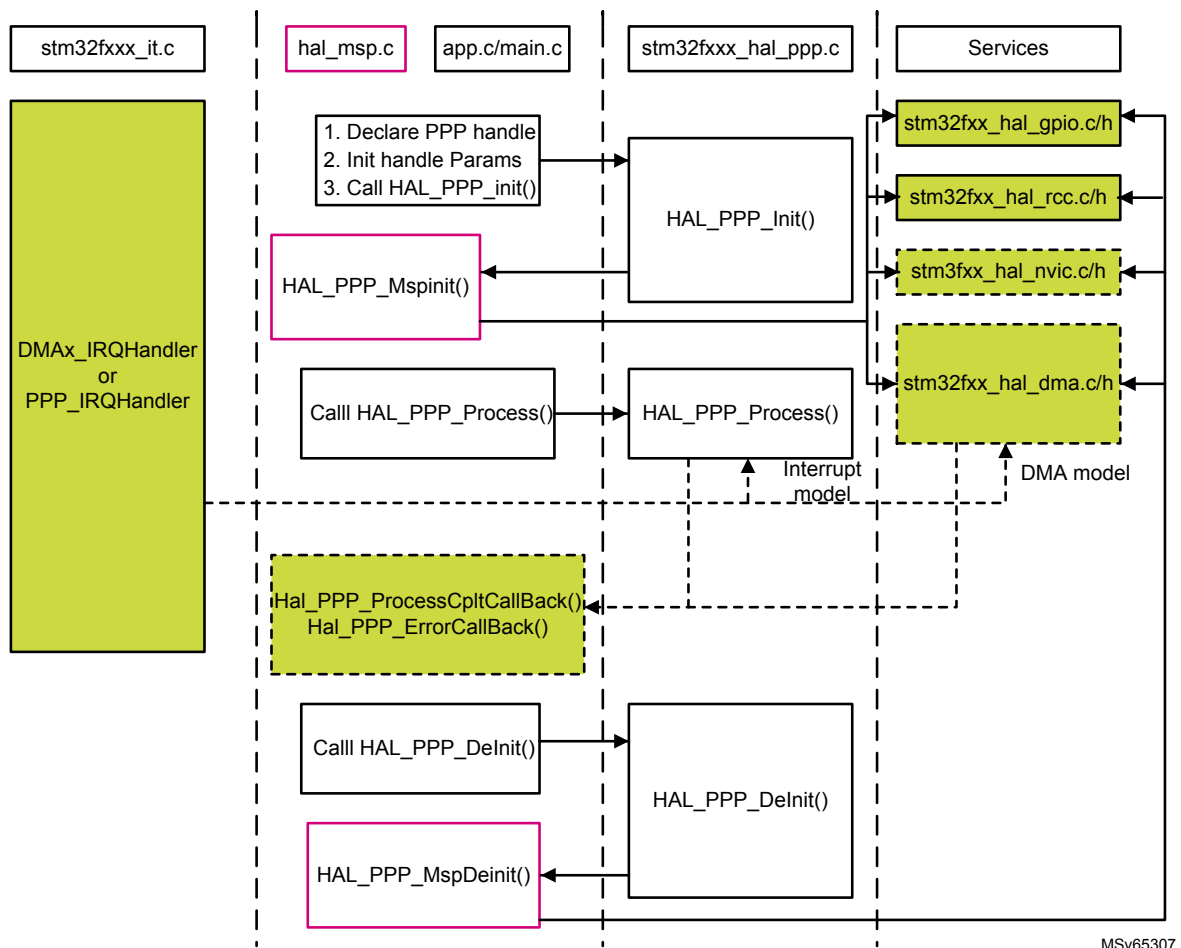
*Note:* DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 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 `stm32f4xx_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
  - set priority grouping to 4 preemption bits
  - 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.

**Note:** *In STM32Cube V1.0 implemented in STM32CubeF2 and STM32CubeF4 first versions, the SysTick timer is used as default timebase. This has been modified to allow implementing user-defined timebases (such as a general-purpose timer), keeping in mind that the timebase duration must be kept at 1 ms since all PPP\_TIMEOUT\_VALUES are defined and handled in milliseconds. This enhancement is implemented in STM32Cube V1.1 that is deployed starting from STM32CubeL0/F0/F3 and later. This modification is backward compatible with STM32Cube V1.0 implementation. Functions affecting timebase configurations are declared as `__Weak` to allow different implementations in the user file.*

### 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.

```
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);
  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
  RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5); }
```

### 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 *stm32f4xx\_hal\_msp.c* file in the user folders. An *stm32f4xx\_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.

*stm32f4xx\_hal\_msp.c* file contains the following functions:

**Table 14. MSP functions**

Routine	Description
<b>void HAL_MspInit()</b>	Global MSP initialization routine
<b>void HAL_MspDeInit()</b>	Global MSP de-initialization routine
<b>void HAL_PPP_MspInit()</b>	PPP MSP initialization routine
<b>void HAL_PPP_MspDeInit()</b>	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_t Size, uint32_t Timeout)
{
  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 *stm32f4xx\_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 = USART3;
  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)
{
}

```

*stm32f4xx\_it.c* file:

```
extern UART_HandleTypeDef UartHandle;
void USART3_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 *stm32f4xx\_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 = USART3;
  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 = USART3;
  HAL_UART_Init(&UartHandle);
  HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
  while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *pUART)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *pUART)
{
}
```

*stm32f4xx\_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) <sup>(1)</sup>	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

1. HAL\_MAX\_DELAY is defined in the *stm32f4xx\_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 */
PPP->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); /* retrieve 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 `stm32f4xx_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 `stm32f4xx_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)
  {
  }
}
```

**Attention:** *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
<i>stm32f4xx_ll_bus.h</i>	This is the h-source file for core bus control and peripheral clock activation and deactivation <i>Example: LL_AHB2_GRP1_EnableClock</i>
<i>stm32f4xx_ll_ppp.h/.c</i>	<i>stm32f4xx_ll_ppp.c</i> provides peripheral initialization functions such as <i>LL_PPP_Init()</i> , <i>LL_PPP_StructInit()</i> , <i>LL_PPP_DeInit()</i> . All the other APIs are defined within <i>stm32f4xx_ll_ppp.h</i> file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the <i>stm32f4xx_ll_ppp.h</i> file.
<i>stm32f4xx_ll_cortex.h</i>	Cortex-M related register operation APIs including the SysTick, Low power (such as <i>LL_SYSTICK_XXXXX</i> and <i>LL_LPM_XXXXX</i> "Low Power Mode")
<i>stm32f4xx_ll_utils.h/.c</i>	This file covers the generic APIs: <ul style="list-style-type: none"> <li>• Read of device unique ID and electronic signature</li> <li>• Timebase and delay management</li> <li>• System clock configuration.</li> </ul>
<i>stm32f4xx_ll_system.h</i>	System related operations. <i>Example: LL_SYSCFG_XXX, LL_DBGMCU_XXX and LL_FLASH_XXX and LL_VREFBUF_XXX</i>
<i>stm32_assert_template.h</i>	Template file allowing to define the <i>assert_param</i> 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 <i>stm32_assert.h</i> .

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

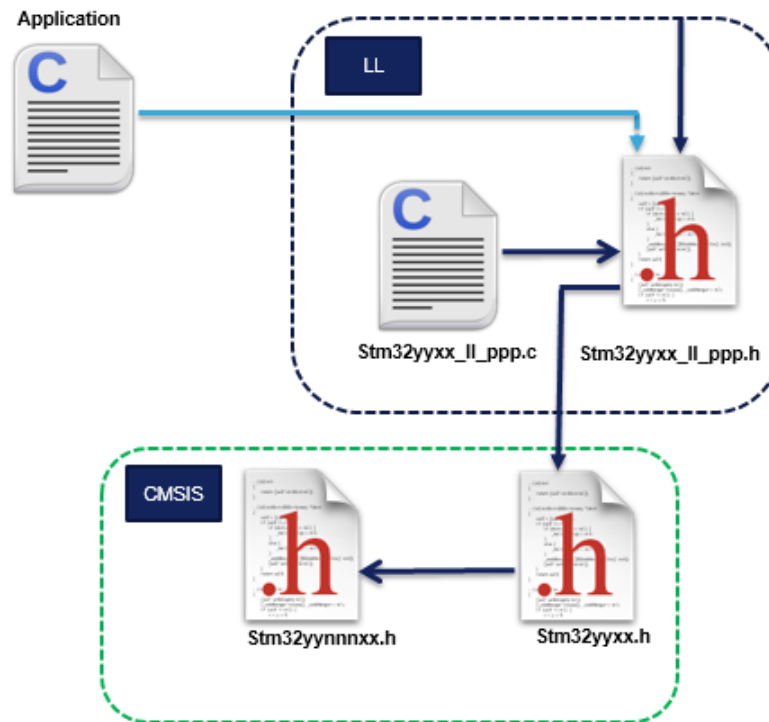
The low-layer files are located in the same HAL driver folder.

**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 `stm32f4xx_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	<i>ErrorStatus</i>	<ul style="list-style-type: none"> <li><i>PPP_TypeDef* PPPx</i></li> <li><i>LL_PPP_InitTypeDef* PPP_InitStruct</i></li> </ul>	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	<i>void</i>	<ul style="list-style-type: none"> <li><i>LL_PPP_InitTypeDef* PPP_InitStruct</i></li> </ul>	Fills each PPP_InitStruct member with its default value. Example: LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	<i>ErrorStatus</i>	<ul style="list-style-type: none"> <li><i>PPP_TypeDef* PPPx</i></li> </ul>	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	<i>ErrorStatus</i>	<ul style="list-style-type: none"> <li><i>PPP_TypeDef* PPPx</i></li> <li><i>LL_PPP{CATEGORY}_InitTypeDef* PPP{CATEGORY}_InitStruct</i></li> </ul>	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	<i>void</i>	<i>LL_PPP{CATEGORY}_InitTypeDef* PPP{CATEGORY}_InitStruct</i>	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	<i>ErrorStatus</i>	<ul style="list-style-type: none"> <li><i>PPP_TypeDef* PPPx</i></li> <li><i>LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct</i></li> </ul>	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	<i>void</i>	<i>LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct</i>	Fills each PPP_CommonInitStruct member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	<i>ErrorStatus</i>	<ul style="list-style-type: none"> <li><i>PPP_TypeDef* PPPx</i></li> <li><i>LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct</i></li> </ul>	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

Functions	Return Type	Parameters	Examples
LL_PPP_ClockStructInit	void	LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct	Fills each <i>PPP_ClockInitStruct</i> 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	<ul style="list-style-type: none"> <li>• LL_RCC_IsActiveFlag_LSIRDY</li> <li>• LL_RCC_IsActiveFlag_FWRST()</li> <li>• LL_ADC_ClearFlag_EOC(ADC1)</li> <li>• LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)</li> </ul>

**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
<code>LL_BUS_GRPx_ActionClock{Mode}</code>	<ul style="list-style-type: none"> <li><code>LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA LL_AHB2_GRP1_PERIPH_GPIOB)</code></li> <li><code>LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)</code></li> </ul>

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
<code>LL_PPP{CATEGORY}_Action{Item}</code> <code>LL_PPP{CATEGORY}_IsItemAction</code>	<ul style="list-style-type: none"> <li><code>LL_ADC_Enable ()</code></li> <li><code>LL_ADC_StartCalibration();</code></li> <li><code>LL_ADC_IsCalibrationOnGoing;</code></li> <li><code>LL_RCC_HSI_Enable ()</code></li> <li><code>LL_RCC_HSI_IsReady()</code></li> </ul>

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

**Table 23. Peripheral configuration management**

Name	Examples
<code>LL_PPP{CATEGORY}_Set{ or Get}ConfigItem</code>	<code>LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)</code>

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

**Table 24. Peripheral register management**

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

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-layer 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 `stm32f4xx_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 [STM32CubeF4](#) 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 `stm32f4` firmware package (refer to `Examples_MIX` projects).

- Note:*
1. When the HAL `Init/DelInit` 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:

- Initializes the Flash interface the NVIC allocation and initial clock configuration. It initializes the systick also when timeout is needed and the backup domain when enabled.
- De-Initializes 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
- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

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()*
- *HAL\_DBGMCU\_EnableDBGSleepMode()*
- *HAL\_DBGMCU\_DisableDBGSleepMode()*
- *HAL\_DBGMCU\_EnableDBGStopMode()*
- *HAL\_DBGMCU\_DisableDBGStopMode()*
- *HAL\_DBGMCU\_EnableDBGStandbyMode()*
- *HAL\_DBGMCU\_DisableDBGStandbyMode()*
- *HAL\_EnableCompensationCell()*
- *HAL\_DisableCompensationCell()*
- *HAL\_GetUIDw0()*
- *HAL\_GetUIDw1()*
- *HAL\_GetUIDw2()*
- *HAL\_EnableMemorySwappingBank()*
- *HAL\_DisableMemorySwappingBank()*

#### 6.1.4 Detailed description of functions

##### HAL\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_Init (void )**

###### Function description

This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it performs the following: Configure the Flash prefetch, instruction and Data caches.

###### Return values

- **HAL:** status

###### Notes

- SysTick is used as time base for the HAL\_Delay() function, the application need to ensure that the SysTick time base is always set to 1 millisecond to have correct HAL operation.

##### HAL\_DeInit

###### Function name

**HAL\_StatusTypeDef HAL\_DeInit (void )**

###### Function description

This function de-Initializes common part of the HAL and stops the systick.

###### Return values

- **HAL:** status

### HAL\_Msplnit

#### Function name

**void HAL\_Msplnit (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.

#### 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 (HAL\_TickFreqTypeDef Freq)**

**Function description**

Set new tick Freq.

**Return values**

- **Status:**



### HAL\_GetTickFreq

#### Function name

**HAL\_TickFreqTypeDef HAL\_GetTickFreq (void )**

#### Function description

Return tick frequency.

#### Return values

- **tick:** period in Hz

### HAL\_SuspendTick

#### Function name

**void HAL\_SuspendTick (void )**

#### Function description

Suspend 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 STOP mode.

#### Return values

- **None:**

### **HAL\_DBGMCU\_DisableDBGStopMode**

**Function name**

**void HAL\_DBGMCU\_DisableDBGStopMode (void )**

**Function description**

Disable the Debug Module during STOP mode.

**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\_EnableCompensationCell**

**Function name**

**void HAL\_EnableCompensationCell (void )**

**Function description**

Enables the I/O Compensation Cell.

**Return values**

- **None:**

**Notes**

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

### **HAL\_DisableCompensationCell**

**Function name**

**void HAL\_DisableCompensationCell (void )**

**Function description**

Power-down the I/O Compensation Cell.

**Return values**

- **None:**

**Notes**

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

**HAL\_GetUIDw0**

**Function name**

`uint32_t HAL_GetUIDw0 (void )`

**Function description**

Returns first word of the unique device identifier (UID based on 96 bits)

**Return values**

- **Device:** identifier

**HAL\_GetUIDw1**

**Function name**

`uint32_t HAL_GetUIDw1 (void )`

**Function description**

Returns second word of the unique device identifier (UID based on 96 bits)

**Return values**

- **Device:** identifier

**HAL\_GetUIDw2**

**Function name**

`uint32_t HAL_GetUIDw2 (void )`

**Function description**

Returns third word of the unique device identifier (UID based on 96 bits)

**Return values**

- **Device:** identifier

**HAL\_EnableMemorySwappingBank**

**Function name**

`void HAL_EnableMemorySwappingBank (void )`

**Function description**

Enables the Internal FLASH Bank Swapping.

**Return values**

- **None:**

**Notes**

- This function can be used only for STM32F42xxx/43xxx/469xx/479xx devices.
- Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08100000 (and aliased at 0x00100000)

**HAL\_DisableMemorySwappingBank**

**Function name**

`void HAL_DisableMemorySwappingBank (void )`

### Function description

Disables the Internal FLASH Bank Swapping.

### Return values

- **None:**

### Notes

- This function can be used only for STM32F42xxx/43xxx/469xx/479xx devices.
- The default state : Flash Bank1 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank2 mapped at 0x08100000 (and aliased at 0x00100000)

## 6.2 HAL Firmware driver defines

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

### 6.2.1 HAL

HAL

#### *HAL Exported Macros*

`__HAL_DBGMCU_FREEZE_TIM2`

`__HAL_DBGMCU_FREEZE_TIM3`

`__HAL_DBGMCU_FREEZE_TIM4`

`__HAL_DBGMCU_FREEZE_TIM5`

`__HAL_DBGMCU_FREEZE_TIM6`

`__HAL_DBGMCU_FREEZE_TIM7`

`__HAL_DBGMCU_FREEZE_TIM12`

`__HAL_DBGMCU_FREEZE_TIM13`

`__HAL_DBGMCU_FREEZE_TIM14`

`__HAL_DBGMCU_FREEZE_RTC`

`__HAL_DBGMCU_FREEZE_WWDG`

`__HAL_DBGMCU_FREEZE_IWDG`

`__HAL_DBGMCU_FREEZE_I2C1_TIMEOUT`

`__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT`

`__HAL_DBGMCU_FREEZE_I2C3_TIMEOUT`

`__HAL_DBGMCU_FREEZE_CAN1`

`__HAL_DBGMCU_FREEZE_CAN2`

`__HAL_DBGMCU_FREEZE_TIM1`

`__HAL_DBGMCU_FREEZE_TIM8`

---

```
__HAL_DBGMCU_FREEZE_TIM9
__HAL_DBGMCU_FREEZE_TIM10
__HAL_DBGMCU_FREEZE_TIM11
__HAL_DBGMCU_UNFREEZE_TIM2
__HAL_DBGMCU_UNFREEZE_TIM3
__HAL_DBGMCU_UNFREEZE_TIM4
__HAL_DBGMCU_UNFREEZE_TIM5
__HAL_DBGMCU_UNFREEZE_TIM6
__HAL_DBGMCU_UNFREEZE_TIM7
__HAL_DBGMCU_UNFREEZE_TIM12
__HAL_DBGMCU_UNFREEZE_TIM13
__HAL_DBGMCU_UNFREEZE_TIM14
__HAL_DBGMCU_UNFREEZE_RTC
__HAL_DBGMCU_UNFREEZE_WWDG
__HAL_DBGMCU_UNFREEZE_IWDG
__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT
__HAL_DBGMCU_UNFREEZE_CAN1
__HAL_DBGMCU_UNFREEZE_CAN2
__HAL_DBGMCU_UNFREEZE_TIM1
__HAL_DBGMCU_UNFREEZE_TIM8
__HAL_DBGMCU_UNFREEZE_TIM9
__HAL_DBGMCU_UNFREEZE_TIM10
__HAL_DBGMCU_UNFREEZE_TIM11
__HAL_SYSCFG_REMAPMEMORY_FLASH
__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH
__HAL_SYSCFG_REMAPMEMORY_SRAM
```

\_\_HAL\_SYSCFG\_REMAPMEMORY\_FMC

\_\_HAL\_SYSCFG\_REMAPMEMORY\_FMC\_SDRAM

## 7 HAL ADC Generic Driver

### 7.1 ADC Firmware driver registers structures

#### 7.1.1 ADC\_InitTypeDef

*ADC\_InitTypeDef* is defined in the `stm32f4xx_hal_adc.h`

##### Data Fields

- *uint32\_t* **ClockPrescaler**
- *uint32\_t* **Resolution**
- *uint32\_t* **DataAlign**
- *uint32\_t* **ScanConvMode**
- *uint32\_t* **EOCSelection**
- **FunctionalState** *ContinuousConvMode*
- *uint32\_t* **NbrOfConversion**
- **FunctionalState** *DiscontinuousConvMode*
- *uint32\_t* **NbrOfDiscConversion**
- *uint32\_t* **ExternalTrigConv**
- *uint32\_t* **ExternalTrigConvEdge**
- **FunctionalState** *DMAContinuousRequests*

##### Field Documentation

- *uint32\_t* **ADC\_InitTypeDef::ClockPrescaler**  
Select ADC clock prescaler. The clock is common for all the ADCs. This parameter can be a value of [ADC\\_ClockPrescaler](#)
- *uint32\_t* **ADC\_InitTypeDef::Resolution**  
Configures the ADC resolution. This parameter can be a value of [ADC\\_Resolution](#)
- *uint32\_t* **ADC\_InitTypeDef::DataAlign**  
Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset application): MSB on register bit 14 and LSB on register bit 3). This parameter can be a value of [ADC\\_Data\\_align](#)
- *uint32\_t* **ADC\_InitTypeDef::ScanConvMode**  
Configures the sequencer of regular and injected groups. 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'/'InjectedNbrOfConversion' and each channel rank). Scan direction is upward: from rank1 to rank 'n'. This parameter can be set to ENABLE or DISABLE
- *uint32\_t* **ADC\_InitTypeDef::EOCSelection**  
Specifies what EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of conversion of each rank or complete sequence. This parameter can be a value of [ADC\\_EOCSelection](#).  
Note: For injected group, end of conversion (flag&IT) is raised only at the end of the sequence.  
Therefore, if end of conversion is set to end of each conversion, injected group should not be used with interruption (`HAL_ADCEX_InjectedStart_IT`) or polling (`HAL_ADCEX_InjectedStart` and `HAL_ADCEX_InjectedPollForConversion`). By the way, polling is still possible since driver will use an estimated timing for end of injected conversion. Note: If overrun feature is intended to be used, use ADC in mode 'interruption' (function `HAL_ADC_Start_IT()`) with parameter `EOCSelection` set to end of each conversion or in mode 'transfer by DMA' (function `HAL_ADC_Start_DMA()`). If overrun feature is intended to be bypassed, use ADC in mode 'polling' or 'interruption' with parameter `EOCSelection` must be set to end of sequence



- **FunctionalState ADC\_InitTypeDef::ContinuousConvMode**  
 Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- **uint32\_t ADC\_InitTypeDef::NbrOfConversion**  
 Specifies the number of ranks that will be converted within the regular group sequencer. To use 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.
- **FunctionalState ADC\_InitTypeDef::DiscontinuousConvMode**  
 Specifies whether the conversions sequence of regular group 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 regular group (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**  
 Selects the external event used to trigger the conversion start of regular group. If set to ADC\_SOFTWARE\_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge by default. This parameter can be a value of [ADC\\_External\\_trigger\\_Source\\_Regular](#)
- **uint32\_t ADC\_InitTypeDef::ExternalTrigConvEdge**  
 Selects the external trigger edge of regular group. If trigger is set to ADC\_SOFTWARE\_START, this parameter is discarded. This parameter can be a value of [ADC\\_External\\_trigger\\_edge\\_Regular](#)
- **FunctionalState ADC\_InitTypeDef::DMAContinuousRequests**  
 Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. 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). This parameter can be set to ENABLE or DISABLE.

### 7.1.2 ADC\_ChannelConfTypeDef

**ADC\_ChannelConfTypeDef** is defined in the `stm32f4xx_hal_adc.h`

#### Data Fields

- **uint32\_t Channel**
- **uint32\_t Rank**
- **uint32\_t SamplingTime**
- **uint32\_t Offset**

#### Field Documentation

- **uint32\_t ADC\_ChannelConfTypeDef::Channel**  
 Specifies the channel to configure into ADC regular group. This parameter can be a value of [ADC\\_channels](#)
- **uint32\_t ADC\_ChannelConfTypeDef::Rank**  
 Specifies the rank in the regular group sequencer. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16

- `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 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits). This parameter can be a value of [ADC\\_sampling\\_times](#)  
 Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to 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, parameters TS\_vrefint, TS\_temp (values rough order: 4us min).
- `uint32_t ADC_ChannelConfTypeDef::Offset`**  
 Reserved for future use, can be set to 0

### 7.1.3 ADC\_AnalogWDGConfTypeDef

**`ADC_AnalogWDGConfTypeDef`** is defined in the `stm32f4xx_hal_adc.h`

#### Data Fields

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

#### Field Documentation

- `uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode`**  
 Configures the ADC analog watchdog mode. This parameter can be a value of [ADC\\_analog\\_watchdog\\_selection](#)
- `uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold`**  
 Configures the ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
- `uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold`**  
 Configures the ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
- `uint32_t ADC_AnalogWDGConfTypeDef::Channel`**  
 Configures ADC channel for the analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel This parameter can be a value of [ADC\\_channels](#)
- `FunctionalState ADC_AnalogWDGConfTypeDef::ITMode`**  
 Specifies whether the analog watchdog is configured is interrupt mode or in polling mode. This parameter can be set to ENABLE or DISABLE
- `uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber`**  
 Reserved for future use, can be set to 0

### 7.1.4 ADC\_HandleTypeDef

**`ADC_HandleTypeDef`** is defined in the `stm32f4xx_hal_adc.h`

#### Data Fields

- `ADC_TypeDef * Instance`**
- `ADC_InitTypeDef Init`**
- `__IO uint32_t NbrOfCurrentConversionRank`**
- `DMA_HandleTypeDef * DMA_Handle`**
- `HAL_LockTypeDef Lock`**
- `__IO uint32_t State`**
- `__IO uint32_t ErrorCode`**

#### Field Documentation

- `ADC_TypeDef* ADC_HandleTypeDef::Instance`**  
 Register base address

- ***ADC\_InitTypeDef ADC\_HandleTypeDef::Init***  
ADC required parameters
- ***\_\_IO uint32\_t ADC\_HandleTypeDef::NbrOfCurrentConversionRank***  
ADC number of current conversion rank
- ***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
- ***\_\_IO uint32\_t ADC\_HandleTypeDef::ErrorCode***  
ADC Error code

## 7.2 ADC Firmware driver API description

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

### 7.2.1 ADC Peripheral features

1. 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.
2. Interrupt generation at the end of conversion, end of injected conversion, and in case of analog watchdog or overrun events
3. Single and continuous conversion modes.
4. Scan mode for automatic conversion of channel 0 to channel x.
5. Data alignment with in-built data coherency.
6. Channel-wise programmable sampling time.
7. External trigger option with configurable polarity for both regular and injected conversion.
8. Dual/Triple mode (on devices with 2 ADCs or more).
9. Configurable DMA data storage in Dual/Triple ADC mode.
10. Configurable delay between conversions in Dual/Triple interleaved mode.
11. ADC conversion type (refer to the datasheets).
12. ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
13. ADC input range: VREF(minus) = VIN = VREF(plus).
14. DMA request generation during regular channel conversion.

### 7.2.2 How to use this driver

1. Initialize the ADC low level resources by implementing the HAL\_ADC\_MspInit():
  - a. Enable the ADC interface clock using `__HAL_RCC_ADC_CLK_ENABLE()`
  - b. ADC pins configuration
    - Enable the clock for the ADC GPIOs using the following function: `__HAL_RCC_GPIOx_CLK_ENABLE()`
    - Configure these ADC pins in analog mode using `HAL_GPIO_Init()`
  - c. In case of using interrupts (e.g. `HAL_ADC_Start_IT()`)
    - Configure the ADC interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the ADC IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In ADC IRQ handler, call `HAL_ADC_IRQHandler()`
  - d. In case of using DMA to control data transfer (e.g. `HAL_ADC_Start_DMA()`)
    - Enable the DMAx interface clock using `__HAL_RCC_DMAx_CLK_ENABLE()`
    - Configure and enable two DMA streams 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.

#### **Configuration of ADC, groups regular/injected, 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 injected group parameters (conversion trigger, sequencer, ..., of injected group) and the channels for injected group parameters (channel number, channel rank into sequencer, ..., into injected group) using function `HAL_ADCEx_InjectedConfigChannel()`.
4. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function `HAL_ADC_AnalogWDGConfig()`.
5. Optionally, for devices with several ADC instances: configure the multimode parameters using function `HAL_ADCEx_MultiModeConfigChannel()`.

#### **Execution of ADC conversions**

1. ADC driver can be used among three modes: polling, interruption, transfer by DMA.

##### **Polling mode IO operation**

- Start the ADC peripheral using `HAL_ADC_Start()`
- Wait for end of conversion using `HAL_ADC_PollForConversion()`, at this stage user can specify the value of timeout according to his end application
- To read the ADC converted values, use the `HAL_ADC_GetValue()` function.
- Stop the ADC peripheral using `HAL_ADC_Stop()`

##### **Interrupt mode IO operation**

- Start the ADC peripheral using `HAL_ADC_Start_IT()`
- Use `HAL_ADC_IRQHandler()` called under `ADC_IRQHandler()` Interrupt subroutine
- At ADC end of conversion `HAL_ADC_ConvCpltCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADC_ConvCpltCallback`
- In case of ADC Error, `HAL_ADC_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADC_ErrorCallback`
- Stop the ADC peripheral using `HAL_ADC_Stop_IT()`

##### **DMA mode IO operation**

- Start the ADC peripheral using `HAL_ADC_Start_DMA()`, at this stage the user specify the length of data to be transferred at each end of conversion

- At The end of data transfer by HAL\_ADC\_ConvCpltCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ConvCpltCallback
- In case of transfer Error, HAL\_ADC\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ErrorCallback
- Stop the ADC peripheral using HAL\_ADC\_Stop\_DMA()

### ADC HAL driver macros list

Below the list of most used macros in ADC HAL driver.

- `__HAL_ADC_ENABLE` : Enable the ADC peripheral
- `__HAL_ADC_DISABLE` : Disable the ADC peripheral
- `__HAL_ADC_ENABLE_IT`: Enable the ADC end of conversion interrupt
- `__HAL_ADC_DISABLE_IT`: Disable the ADC end of conversion interrupt
- `__HAL_ADC_GET_IT_SOURCE`: Check if the specified ADC interrupt source is enabled or disabled
- `__HAL_ADC_CLEAR_FLAG`: Clear the ADC's pending flags
- `__HAL_ADC_GET_FLAG`: Get the selected ADC's flag status
- `ADC_GET_RESOLUTION`: Return resolution bits in CR1 register

*Note:* You can refer to the ADC HAL driver header file for more useful macros

### 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 `__HAL_RCC_ADC_FORCE_RESET()`, `__HAL_RCC_ADC_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:
    - `HAL_RCC_GetOscConfig(&RCC_OscInitStructure);`
    - `RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;`
    - `RCC_OscInitStructure.HSIState = RCC_HSI_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_DisableIRQ(ADCx_IRQn)`
4. Optionally, in case of usage of DMA:
  - Deinitialize the DMA using function `HAL_DMA_DeInit()`.
  - Disable the NVIC for DMA using function `HAL_NVIC_DisableIRQ(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 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC.

This section contains the following APIs:

- [HAL\\_ADC\\_Init\(\)](#)
- [HAL\\_ADC\\_DeInit\(\)](#)
- [HAL\\_ADC\\_MspInit\(\)](#)
- [HAL\\_ADC\\_MspDeInit\(\)](#)

### 7.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular channel.
- Stop conversion of regular channel.
- Start conversion of regular channel and enable interrupt.
- Stop conversion of regular channel and disable interrupt.
- Start conversion of regular channel and enable DMA transfer.

- Stop conversion of regular channel and disable DMA transfer.
- Handle ADC interrupt request.

This section contains the following APIs:

- [\*HAL\\_ADC\\_Start\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\(\)\*](#)
- [\*HAL\\_ADC\\_PollForConversion\(\)\*](#)
- [\*HAL\\_ADC\\_PollForEvent\(\)\*](#)
- [\*HAL\\_ADC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_IRQHandler\(\)\*](#)
- [\*HAL\\_ADC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_DMA\(\)\*](#)
- [\*HAL\\_ADC\\_GetValue\(\)\*](#)
- [\*HAL\\_ADC\\_ConvCpltCallback\(\)\*](#)
- [\*HAL\\_ADC\\_ConvHalfCpltCallback\(\)\*](#)
- [\*HAL\\_ADC\\_LevelOutOfWindowCallback\(\)\*](#)
- [\*HAL\\_ADC\\_ErrorCallback\(\)\*](#)

### 7.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure regular channels.
- Configure injected channels.
- Configure multimode.
- Configure the analog watch dog.

This section contains the following APIs:

- [\*HAL\\_ADC\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_ADC\\_AnalogWDGConfig\(\)\*](#)

### 7.2.6 Peripheral State and errors functions

This subsection provides functions allowing to

- Check the ADC state
- Check the ADC Error

This section contains the following APIs:

- [\*HAL\\_ADC\\_GetState\(\)\*](#)
- [\*HAL\\_ADC\\_GetError\(\)\*](#)

### 7.2.7 Detailed description of functions

#### HAL\_ADC\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_ADC\_Init (ADC\_HandleTypeDef \* hadc)**

##### Function description

Initializes the ADCx peripheral according to the specified parameters in the ADC\_InitStruct and initializes the ADC MSP.

##### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **HAL:** status

### Notes

- This function is used to configure the global features of the ADC ( ClockPrescaler, Resolution, Data Alignment and number of conversion), however, the rest of the configuration parameters are specific to the regular channels group (scan mode activation, continuous mode activation, External trigger source and edge, DMA continuous request after the last transfer and End of conversion selection).

### HAL\_ADC\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_ADC\_DeInit (ADC\_HandleTypeDef \* hadc)**

#### Function description

Deinitializes the ADCx peripheral registers to their default reset values.

#### Parameters

- **hadc:** pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **HAL:** status

### HAL\_ADC\_MspInit

#### Function name

**void HAL\_ADC\_MspInit (ADC\_HandleTypeDef \* hadc)**

#### Function description

Initializes the ADC MSP.

#### Parameters

- **hadc:** pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **None:**

### HAL\_ADC\_MspDeInit

#### Function name

**void HAL\_ADC\_MspDeInit (ADC\_HandleTypeDef \* hadc)**

#### Function description

DeInitializes the ADC MSP.

#### Parameters

- **hadc:** pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **None:**

### HAL\_ADC\_Start

#### Function name

**HAL\_StatusTypeDef HAL\_ADC\_Start (ADC\_HandleTypeDef \* hadc)**



### Function description

Enables ADC and starts conversion of the regular channels.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.

### Return values

- **HAL**: status

### HAL\_ADC\_Stop

### Function name

`HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)`

### Function description

Disables ADC and stop conversion of regular channels.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.

### Return values

- **HAL**: status.

### Notes

- Caution: This function will stop also injected channels.

### HAL\_ADC\_PollForConversion

### Function name

`HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)`

### Function description

Poll for regular conversion complete.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.
- **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.
- 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.

### HAL\_ADC\_PollForEvent

### Function name

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

### Function description

Poll for conversion event.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **EventType**: the ADC event type. This parameter can be one of the following values:
  - ADC\_AWD\_EVENT: ADC Analog watch Dog event.
  - ADC\_OVR\_EVENT: ADC Overrun event.
- **Timeout**: Timeout value in millisecond.

### Return values

- **HAL**: status

### HAL\_ADC\_Start\_IT

### Function name

**HAL\_StatusTypeDef HAL\_ADC\_Start\_IT (ADC\_HandleTypeDef \* hadc)**

### Function description

Enables the interrupt and starts ADC conversion of regular channels.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **HAL**: status.

### HAL\_ADC\_Stop\_IT

### Function name

**HAL\_StatusTypeDef HAL\_ADC\_Stop\_IT (ADC\_HandleTypeDef \* hadc)**

### Function description

Disables the interrupt and stop ADC conversion of regular channels.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **HAL**: status.

### Notes

- Caution: This function will stop also injected channels.

### HAL\_ADC\_IRQHandler

### Function name

**void HAL\_ADC\_IRQHandler (ADC\_HandleTypeDef \* hadc)**

### Function description

Handles ADC interrupt request.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

#### Return values

- **None:**

**HAL\_ADC\_Start\_DMA**

#### Function name

**HAL\_StatusTypeDef HAL\_ADC\_Start\_DMA (ADC\_HandleTypeDef \* hadc, uint32\_t \* pData, uint32\_t Length)**

#### Function description

Enables ADC DMA request after last transfer (Single-ADC mode) and enables ADC peripheral.

#### Parameters

- **hadc:** pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from ADC peripheral to memory.

#### Return values

- **HAL:** status

**HAL\_ADC\_Stop\_DMA**

#### Function name

**HAL\_StatusTypeDef HAL\_ADC\_Stop\_DMA (ADC\_HandleTypeDef \* hadc)**

#### Function description

Disables ADC DMA (Single-ADC mode) and disables ADC peripheral.

#### Parameters

- **hadc:** pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

#### Return values

- **HAL:** status

**HAL\_ADC\_GetValue**

#### Function name

**uint32\_t HAL\_ADC\_GetValue (ADC\_HandleTypeDef \* hadc)**

#### Function description

Gets the converted value from data register of regular channel.

#### Parameters

- **hadc:** pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

#### Return values

- **Converted:** value

**HAL\_ADC\_ConvCpltCallback**

#### Function name

**void HAL\_ADC\_ConvCpltCallback (ADC\_HandleTypeDef \* hadc)**

#### Function description

Regular conversion complete callback in non blocking mode.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.

### Return values

- **None**:

### HAL\_ADC\_ConvHalfCpltCallback

### Function name

**void HAL\_ADC\_ConvHalfCpltCallback (ADC\_HandleTypeDef \* hadc)**

### Function description

Regular conversion half DMA transfer callback in non blocking mode.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.

### Return values

- **None**:

### HAL\_ADC\_LevelOutOfWindowCallback

### Function name

**void HAL\_ADC\_LevelOutOfWindowCallback (ADC\_HandleTypeDef \* hadc)**

### Function description

Analog watchdog callback in non blocking mode.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.

### Return values

- **None**:

### HAL\_ADC\_ErrorCallback

### Function name

**void HAL\_ADC\_ErrorCallback (ADC\_HandleTypeDef \* hadc)**

### Function description

Error ADC callback.

### Parameters

- **hadc**: pointer to a `ADC_HandleTypeDef` structure that contains the configuration information for the specified ADC.

### 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

Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **sConfig**: ADC configuration structure.

### Return values

- **HAL**: status

## HAL\_ADC\_AnalogWDGConfig

### Function name

**HAL\_StatusTypeDef HAL\_ADC\_AnalogWDGConfig (ADC\_HandleTypeDef \* hadc, ADC\_AnalogWDGConfTypeDef \* AnalogWDGConfig)**

### Function description

Configures the analog watchdog.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **AnalogWDGConfig**: pointer to an ADC\_AnalogWDGConfTypeDef structure that contains the configuration information of ADC analog watchdog.

### Return values

- **HAL**: status

### Notes

- 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 state

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **HAL**: state

## HAL\_ADC\_GetError

### Function name

`uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)`

### Function description

Return the ADC error code.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **ADC**: Error Code

## 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 Selection*

`ADC_ANALOGWATCHDOG_SINGLE_REG`

`ADC_ANALOGWATCHDOG_SINGLE_INJEC`

`ADC_ANALOGWATCHDOG_SINGLE_REGINJEC`

`ADC_ANALOGWATCHDOG_ALL_REG`

`ADC_ANALOGWATCHDOG_ALL_INJEC`

`ADC_ANALOGWATCHDOG_ALL_REGINJEC`

`ADC_ANALOGWATCHDOG_NONE`

#### *ADC Common Channels*

`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

ADC\_CHANNEL\_VREFINT

ADC\_CHANNEL\_VBAT

***ADC Channels Type***

ADC\_ALL\_CHANNELS

ADC\_REGULAR\_CHANNELS

reserved for future use

ADC\_INJECTED\_CHANNELS

reserved for future use

***ADC Clock Prescaler***

ADC\_CLOCK\_SYNC\_PCLK\_DIV2

ADC\_CLOCK\_SYNC\_PCLK\_DIV4

ADC\_CLOCK\_SYNC\_PCLK\_DIV6

ADC\_CLOCK\_SYNC\_PCLK\_DIV8

***ADC Data Align***

ADC\_DATAALIGN\_RIGHT

ADC\_DATAALIGN\_LEFT

***ADC Delay Between 2 Sampling Phases***

ADC\_TWOSAMPLINGDELAY\_5CYCLES

ADC\_TWOSAMPLINGDELAY\_6CYCLES

ADC\_TWOSAMPLINGDELAY\_7CYCLES

ADC\_TWOSAMPLINGDELAY\_8CYCLES

ADC\_TWOSAMPLINGDELAY\_9CYCLES

ADC\_TWOSAMPLINGDELAY\_10CYCLES

ADC\_TWOSAMPLINGDELAY\_11CYCLES

ADC\_TWOSAMPLINGDELAY\_12CYCLES

ADC\_TWOSAMPLINGDELAY\_13CYCLES

ADC\_TWOSAMPLINGDELAY\_14CYCLES

ADC\_TWOSAMPLINGDELAY\_15CYCLES

ADC\_TWOSAMPLINGDELAY\_16CYCLES

ADC\_TWOSAMPLINGDELAY\_17CYCLES

ADC\_TWOSAMPLINGDELAY\_18CYCLES

ADC\_TWOSAMPLINGDELAY\_19CYCLES

ADC\_TWOSAMPLINGDELAY\_20CYCLES

***ADC EOC Selection***

ADC\_EOC\_SEQ\_CONV

ADC\_EOC\_SINGLE\_CONV

ADC\_EOC\_SINGLE\_SEQ\_CONV

reserved for future use

***ADC Error Code***

HAL\_ADC\_ERROR\_NONE

No error

HAL\_ADC\_ERROR\_INTERNAL

ADC IP internal error: if problem of clocking, enable/disable, erroneous state

HAL\_ADC\_ERROR\_OVR

Overrun error

HAL\_ADC\_ERROR\_DMA

DMA transfer error

***ADC Event Type***

ADC\_AWD\_EVENT

ADC\_OVR\_EVENT

***ADC Exported Macros***



### `__HAL_ADC_RESET_HANDLE_STATE`

**Description:**

- Reset ADC handle state.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

### `__HAL_ADC_ENABLE`

**Description:**

- Enable the ADC peripheral.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

### `__HAL_ADC_DISABLE`

**Description:**

- Disable the ADC peripheral.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

### `__HAL_ADC_ENABLE_IT`

**Description:**

- Enable the ADC end of conversion interrupt.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__INTERRUPT__`: ADC Interrupt.

**Return value:**

- None

### `__HAL_ADC_DISABLE_IT`

**Description:**

- Disable the ADC end of conversion interrupt.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__INTERRUPT__`: ADC interrupt.

**Return value:**

- None

### **\_\_HAL\_ADC\_GET\_IT\_SOURCE**

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__INTERRUPT__`: specifies the ADC interrupt source to check.

**Return value:**

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

### **\_\_HAL\_ADC\_CLEAR\_FLAG**

**Description:**

- Clear the ADC's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__FLAG__`: ADC flag.

**Return value:**

- None

### **\_\_HAL\_ADC\_GET\_FLAG**

**Description:**

- Get the selected ADC's flag status.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__FLAG__`: ADC flag.

**Return value:**

- None

**ADC Exported Types**

### **HAL\_ADC\_STATE\_RESET**

ADC not yet initialized or disabled

### **HAL\_ADC\_STATE\_READY**

ADC peripheral ready for use

### **HAL\_ADC\_STATE\_BUSY\_INTERNAL**

ADC is busy to 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 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\_INJ\_BUSY**

A conversion on 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\_AWD1**

Out-of-window occurrence of analog watchdog 1

**HAL\_ADC\_STATE\_AWD2**

Not available on STM32F4 device: Out-of-window occurrence of analog watchdog 2

**HAL\_ADC\_STATE\_AWD3**

Not available on STM32F4 device: Out-of-window occurrence of analog watchdog 3

**HAL\_ADC\_STATE\_MULTIMODE\_SLAVE**

Not available on STM32F4 device: ADC in multimode slave state, controlled by another ADC master (  
***ADC External Trigger Edge Regular***)

**ADC\_EXTERNALTRIGCONVEDGE\_NONE**
**ADC\_EXTERNALTRIGCONVEDGE\_RISING**
**ADC\_EXTERNALTRIGCONVEDGE\_FALLING**
**ADC\_EXTERNALTRIGCONVEDGE\_RISINGFALLING**

***ADC External Trigger Source Regular***

**ADC\_EXTERNALTRIGCONV\_T1\_CC1**
**ADC\_EXTERNALTRIGCONV\_T1\_CC2**
**ADC\_EXTERNALTRIGCONV\_T1\_CC3**
**ADC\_EXTERNALTRIGCONV\_T2\_CC2**
**ADC\_EXTERNALTRIGCONV\_T2\_CC3**
**ADC\_EXTERNALTRIGCONV\_T2\_CC4**
**ADC\_EXTERNALTRIGCONV\_T2\_TRGO**
**ADC\_EXTERNALTRIGCONV\_T3\_CC1**
**ADC\_EXTERNALTRIGCONV\_T3\_TRGO**
**ADC\_EXTERNALTRIGCONV\_T4\_CC4**
**ADC\_EXTERNALTRIGCONV\_T5\_CC1**

ADC\_EXTERNALTRIGCONV\_T5\_CC2

ADC\_EXTERNALTRIGCONV\_T5\_CC3

ADC\_EXTERNALTRIGCONV\_T8\_CC1

ADC\_EXTERNALTRIGCONV\_T8\_TRGO

ADC\_EXTERNALTRIGCONV\_Ext\_IT11

ADC\_SOFTWARE\_START

***ADC Flags Definition***

ADC\_FLAG\_AWD

ADC\_FLAG\_EOC

ADC\_FLAG\_JEOC

ADC\_FLAG\_JSTRT

ADC\_FLAG\_STRT

ADC\_FLAG\_OVR

***ADC Interrupts Definition***

ADC\_IT\_EOC

ADC\_IT\_AWD

ADC\_IT\_JEOC

ADC\_IT\_OVR

***ADC Resolution***

ADC\_RESOLUTION\_12B

ADC\_RESOLUTION\_10B

ADC\_RESOLUTION\_8B

ADC\_RESOLUTION\_6B

***ADC Sampling Times***

ADC\_SAMPLETIME\_3CYCLES

ADC\_SAMPLETIME\_15CYCLES

ADC\_SAMPLETIME\_28CYCLES

ADC\_SAMPLETIME\_56CYCLES

ADC\_SAMPLETIME\_84CYCLES

ADC\_SAMPLETIME\_112CYCLES

ADC\_SAMPLETIME\_144CYCLES

ADC\_SAMPLETIME\_480CYCLES

## 8 HAL ADC Extension Driver

### 8.1 ADCEX Firmware driver registers structures

#### 8.1.1 ADC\_InjectionConfTypeDef

*ADC\_InjectionConfTypeDef* is defined in the `stm32f4xx_hal_adc_ex.h`

##### Data Fields

- *uint32\_t InjectedChannel*
- *uint32\_t InjectedRank*
- *uint32\_t InjectedSamplingTime*
- *uint32\_t InjectedOffset*
- *uint32\_t InjectedNbrOfConversion*
- *FunctionalState InjectedDiscontinuousConvMode*
- *FunctionalState AutoInjectedConv*
- *uint32\_t ExternalTrigInjecConv*
- *uint32\_t ExternalTrigInjecConvEdge*

##### Field Documentation

- *uint32\_t ADC\_InjectionConfTypeDef::InjectedChannel*  
Selection of ADC channel to configure This parameter can be a value of [ADC\\_channels](#) Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability.
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedRank*  
Rank in the injected group sequencer This parameter must be a value of [ADCEX\\_injected\\_rank](#) Note: In case of need 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 can be 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 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits). This parameter can be a value of [ADC\\_sampling\\_times](#) Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to 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, parameters TS\_vrefint, TS\_temp (values rough order: 4us min).
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedOffset*  
Defines the offset to be subtracted from the raw converted data (for channels set on injected group only). 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.
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedNbrOfConversion*  
Specifies the number of ranks that will be converted within the injected group 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 injected group 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. Note: For injected group, number of discontinuous ranks increment is fixed to one-by-one. 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 automatic injected group 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\_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.
- 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. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of **ADCEX\_External\_trigger\_Source\_Injected** 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 behaviour in case of another parameter update on the fly) 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 **ADCEX\_External\_trigger\_edge\_Injected**. If trigger 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.

### 8.1.2 ADC\_MultiModeTypeDef

**ADC\_MultiModeTypeDef** is defined in the stm32f4xx\_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 multi mode. This parameter can be a value of **ADCEX\_Common\_mode**
- uint32\_t ADC\_MultiModeTypeDef::DMAAccessMode**  
 Configures the Direct memory access mode for multi ADC mode. This parameter can be a value of **ADCEX\_Direct\_memory\_access\_mode\_for\_multi\_mode**
- uint32\_t ADC\_MultiModeTypeDef::TwoSamplingDelay**  
 Configures the Delay between 2 sampling phases. This parameter can be a value of **ADC\_delay\_between\_2\_sampling\_phases**

## 8.2 ADCEX Firmware driver API description

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

## 8.2.1 How to use this driver

1. Initialize the ADC low level resources by implementing the HAL\_ADC\_MspInit():
  - a. Enable the ADC interface clock using `__HAL_RCC_ADC_CLK_ENABLE()`
  - b. ADC pins configuration
    - Enable the clock for the ADC GPIOs using the following function: `__HAL_RCC_GPIOx_CLK_ENABLE()`
    - Configure these ADC pins in analog mode using `HAL_GPIO_Init()`
  - c. In case of using interrupts (e.g. `HAL_ADC_Start_IT()`)
    - Configure the ADC interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the ADC IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In ADC IRQ handler, call `HAL_ADC_IRQHandler()`
  - d. In case of using DMA to control data transfer (e.g. `HAL_ADC_Start_DMA()`)
    - Enable the DMAx interface clock using `__HAL_RCC_DMAx_CLK_ENABLE()`
    - Configure and enable two DMA streams stream for managing data transfer from peripheral to memory (output stream)
    - Associate the initialized DMA handle to the ADC 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.
2. Configure the ADC Prescaler, conversion resolution and data alignment using the `HAL_ADC_Init()` function.
3. Configure the ADC Injected channels group features, use `HAL_ADC_Init()` and `HAL_ADC_ConfigChannel()` functions.
4. Three operation modes are available within this driver:

### Polling mode IO operation

- Start the ADC peripheral using `HAL_ADCEx_InjectedStart()`
- Wait for end of conversion using `HAL_ADC_PollForConversion()`, at this stage user can specify the value of timeout according to his end application
- To read the ADC converted values, use the `HAL_ADCEx_InjectedGetValue()` function.
- Stop the ADC peripheral using `HAL_ADCEx_InjectedStop()`

### Interrupt mode IO operation

- Start the ADC peripheral using `HAL_ADCEx_InjectedStart_IT()`
- Use `HAL_ADC_IRQHandler()` called under `ADC_IRQHandler()` Interrupt subroutine
- At ADC end of conversion `HAL_ADCEx_InjectedConvCpltCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADCEx_InjectedConvCpltCallback`
- In case of ADC Error, `HAL_ADCEx_InjectedErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADCEx_InjectedErrorCallback`
- Stop the ADC peripheral using `HAL_ADCEx_InjectedStop_IT()`

### Multi mode ADCs Regular channels configuration

- Select the Multi mode ADC regular channels features (dual or triple mode) and configure the DMA mode using `HAL_ADCEx_MultiModeConfigChannel()` functions.
- Start the ADC peripheral using `HAL_ADCEx_MultiModeStart_DMA()`, at this stage the user specify the length of data to be transferred at each end of conversion
- Read the ADCs converted values using the `HAL_ADCEx_MultiModeGetValue()` function.

## 8.2.2 Extended features functions

This section provides functions allowing to:

- Start conversion of injected channel.
- Stop conversion of injected channel.
- Start multimode and enable DMA transfer.



- Stop multimode and disable DMA transfer.
- Get result of injected channel conversion.
- Get result of multimode conversion.
- Configure injected channels.
- Configure multimode.

This section contains the following APIs:

- *HAL\_ADCEX\_InjectedStart()*
- *HAL\_ADCEX\_InjectedStart\_IT()*
- *HAL\_ADCEX\_InjectedStop()*
- *HAL\_ADCEX\_InjectedPollForConversion()*
- *HAL\_ADCEX\_InjectedStop\_IT()*
- *HAL\_ADCEX\_InjectedGetValue()*
- *HAL\_ADCEX\_MultiModeStart\_DMA()*
- *HAL\_ADCEX\_MultiModeStop\_DMA()*
- *HAL\_ADCEX\_MultiModeGetValue()*
- *HAL\_ADCEX\_InjectedConvCpltCallback()*
- *HAL\_ADCEX\_InjectedConfigChannel()*
- *HAL\_ADCEX\_MultiModeConfigChannel()*

### 8.2.3 Detailed description of functions

#### HAL\_ADCEX\_InjectedStart

##### Function name

**HAL\_StatusTypeDef HAL\_ADCEX\_InjectedStart (ADC\_HandleTypeDef \* hadc)**

##### Function description

Enables the selected ADC software start conversion of the injected channels.

##### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

##### Return values

- **HAL**: status

#### 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

- **None**:

##### 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 auto-injection mode, HAL\_ADC\_Stop must be used.

### HAL\_ADCEX\_InjectedPollForConversion

#### Function name

**HAL\_StatusTypeDef HAL\_ADCEX\_InjectedPollForConversion (ADC\_HandleTypeDef \* hadc, uint32\_t Timeout)**

#### Function description

Poll for injected conversion complete.

#### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **Timeout**: Timeout value in millisecond.

#### Return values

- **HAL**: status

### HAL\_ADCEX\_InjectedStart\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_ADCEX\_InjectedStart\_IT (ADC\_HandleTypeDef \* hadc)**

#### Function description

Enables the interrupt and starts ADC conversion of injected channels.

#### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

#### Return values

- **HAL**: status.

### 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

- **None**:

#### 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.

### HAL\_ADCEX\_InjectedGetValue

#### Function name

**uint32\_t HAL\_ADCEX\_InjectedGetValue (ADC\_HandleTypeDef \* hadc, uint32\_t InjectedRank)**

### Function description

Gets the converted value from data register of injected channel.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **InjectedRank**: the ADC injected rank. This parameter can be one of the following values:
  - ADC\_INJECTED\_RANK\_1: Injected Channel1 selected
  - ADC\_INJECTED\_RANK\_2: Injected Channel2 selected
  - ADC\_INJECTED\_RANK\_3: Injected Channel3 selected
  - ADC\_INJECTED\_RANK\_4: Injected Channel4 selected

### Return values

- **None**:

### HAL\_ADCEx\_MultiModeStart\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_ADCEx\_MultiModeStart\_DMA (ADC\_HandleTypeDef \* hadc, uint32\_t \* pData, uint32\_t Length)**

### Function description

Enables ADC DMA request after last transfer (Multi-ADC mode) and enables ADC peripheral.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **pData**: Pointer to buffer in which transferred from ADC peripheral to memory will be stored.
- **Length**: The length of data to be transferred from ADC peripheral to memory.

### Return values

- **HAL**: status

### Notes

- Caution: This function must be used only with the ADC master.

### HAL\_ADCEx\_MultiModeStop\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_ADCEx\_MultiModeStop\_DMA (ADC\_HandleTypeDef \* hadc)**

### Function description

Disables ADC DMA (multi-ADC mode) and disables ADC peripheral.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **HAL**: status

### HAL\_ADCEx\_MultiModeGetValue

### Function name

**uint32\_t HAL\_ADCEx\_MultiModeGetValue (ADC\_HandleTypeDef \* hadc)**

### Function description

Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **The**: converted data value.

### HAL\_ADCEx\_InjectedConvCpltCallback

### Function name

**void HAL\_ADCEx\_InjectedConvCpltCallback (ADC\_HandleTypeDef \* hadc)**

### Function description

Injected conversion complete callback in non blocking mode.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

### Return values

- **None**:

### HAL\_ADCEx\_InjectedConfigChannel

### Function name

**HAL\_StatusTypeDef HAL\_ADCEx\_InjectedConfigChannel (ADC\_HandleTypeDef \* hadc, ADC\_InjectionConfTypeDef \* sConfigInjected)**

### Function description

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **sConfigInjected**: ADC configuration structure for injected channel.

### Return values

- **None**:

### HAL\_ADCEx\_MultiModeConfigChannel

### Function name

**HAL\_StatusTypeDef HAL\_ADCEx\_MultiModeConfigChannel (ADC\_HandleTypeDef \* hadc, ADC\_MultiModeTypeDef \* multimode)**

### Function description

Configures the ADC multi-mode.

### Parameters

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **multimode**: pointer to an ADC\_MultiModeTypeDef structure that contains the configuration information for multimode.

**Return values**

- **HAL:** status

### 8.3 ADCEX Firmware driver defines

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

#### 8.3.1 ADCEX

ADCEX

***ADC Specific Channels***

**ADC\_CHANNEL\_DIFFERENCIATION\_TEMPSENSOR\_VBAT**

**ADC\_CHANNEL\_TEMPSENSOR**

***ADC Common Mode***

**ADC\_MODE\_INDEPENDENT**

**ADC\_DUALMODE\_REGSIMULT\_INJECSIMULT**

**ADC\_DUALMODE\_REGSIMULT\_ALTERTRIG**

**ADC\_DUALMODE\_INJECSIMULT**

**ADC\_DUALMODE\_REGSIMULT**

**ADC\_DUALMODE\_INTERL**

**ADC\_DUALMODE\_ALTERTRIG**

**ADC\_TRIPLEMODE\_REGSIMULT\_INJECSIMULT**

**ADC\_TRIPLEMODE\_REGSIMULT\_AlterTrig**

**ADC\_TRIPLEMODE\_INJECSIMULT**

**ADC\_TRIPLEMODE\_REGSIMULT**

**ADC\_TRIPLEMODE\_INTERL**

**ADC\_TRIPLEMODE\_ALTERTRIG**

***ADC Direct Memory Access Mode For Multi Mode***

**ADC\_DMAACCESSMODE\_DISABLED**

DMA mode disabled

**ADC\_DMAACCESSMODE\_1**

DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)

**ADC\_DMAACCESSMODE\_2**

DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)

**ADC\_DMAACCESSMODE\_3**

DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2)

***ADC External Trigger Edge Injected***

ADC\_EXTERNALTRIGINJECCONVEDGE\_NONE

ADC\_EXTERNALTRIGINJECCONVEDGE\_RISING

ADC\_EXTERNALTRIGINJECCONVEDGE\_FALLING

ADC\_EXTERNALTRIGINJECCONVEDGE\_RISINGFALLING

***ADC External Trigger Source Injected***

ADC\_EXTERNALTRIGINJECCONV\_T1\_CC4

ADC\_EXTERNALTRIGINJECCONV\_T1\_TRGO

ADC\_EXTERNALTRIGINJECCONV\_T2\_CC1

ADC\_EXTERNALTRIGINJECCONV\_T2\_TRGO

ADC\_EXTERNALTRIGINJECCONV\_T3\_CC2

ADC\_EXTERNALTRIGINJECCONV\_T3\_CC4

ADC\_EXTERNALTRIGINJECCONV\_T4\_CC1

ADC\_EXTERNALTRIGINJECCONV\_T4\_CC2

ADC\_EXTERNALTRIGINJECCONV\_T4\_CC3

ADC\_EXTERNALTRIGINJECCONV\_T4\_TRGO

ADC\_EXTERNALTRIGINJECCONV\_T5\_CC4

ADC\_EXTERNALTRIGINJECCONV\_T5\_TRGO

ADC\_EXTERNALTRIGINJECCONV\_T8\_CC2

ADC\_EXTERNALTRIGINJECCONV\_T8\_CC3

ADC\_EXTERNALTRIGINJECCONV\_T8\_CC4

ADC\_EXTERNALTRIGINJECCONV\_EXT\_IT15

ADC\_INJECTED\_SOFTWARE\_START

***ADC Injected Rank***

ADC\_INJECTED\_RANK\_1

ADC\_INJECTED\_RANK\_2

ADC\_INJECTED\_RANK\_3

ADC\_INJECTED\_RANK\_4

## 9 HAL CAN Generic Driver

### 9.1 CAN Firmware driver registers structures

#### 9.1.1 CAN\_InitTypeDef

*CAN\_InitTypeDef* is defined in the `stm32f4xx_hal_can.h`

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Mode*
- *uint32\_t SyncJumpWidth*
- *uint32\_t TimeSeg1*
- *uint32\_t TimeSeg2*
- *FunctionalState TimeTriggeredMode*
- *FunctionalState AutoBusOff*
- *FunctionalState AutoWakeUp*
- *FunctionalState AutoRetransmission*
- *FunctionalState ReceiveFifoLocked*
- *FunctionalState TransmitFifoPriority*

##### Field Documentation

- *uint32\_t CAN\_InitTypeDef::Prescaler*  
Specifies the length of a time quantum. This parameter must be a number between `Min_Data = 1` and `Max_Data = 1024`.
- *uint32\_t CAN\_InitTypeDef::Mode*  
Specifies the CAN operating mode. This parameter can be a value of [CAN\\_operating\\_mode](#)
- *uint32\_t CAN\_InitTypeDef::SyncJumpWidth*  
Specifies the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter can be a value of [CAN\\_synchronisation\\_jump\\_width](#)
- *uint32\_t CAN\_InitTypeDef::TimeSeg1*  
Specifies the number of time quanta in Bit Segment 1. This parameter can be a value of [CAN\\_time\\_quantum\\_in\\_bit\\_segment\\_1](#)
- *uint32\_t CAN\_InitTypeDef::TimeSeg2*  
Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of [CAN\\_time\\_quantum\\_in\\_bit\\_segment\\_2](#)
- *FunctionalState CAN\_InitTypeDef::TimeTriggeredMode*  
Enable or disable the time triggered communication mode. This parameter can be set to ENABLE or DISABLE.
- *FunctionalState CAN\_InitTypeDef::AutoBusOff*  
Enable or disable the automatic bus-off management. This parameter can be set to ENABLE or DISABLE.
- *FunctionalState CAN\_InitTypeDef::AutoWakeUp*  
Enable or disable the automatic wake-up mode. This parameter can be set to ENABLE or DISABLE.
- *FunctionalState CAN\_InitTypeDef::AutoRetransmission*  
Enable or disable the non-automatic retransmission mode. This parameter can be set to ENABLE or DISABLE.
- *FunctionalState CAN\_InitTypeDef::ReceiveFifoLocked*  
Enable or disable the Receive FIFO Locked mode. This parameter can be set to ENABLE or DISABLE.
- *FunctionalState CAN\_InitTypeDef::TransmitFifoPriority*  
Enable or disable the transmit FIFO priority. This parameter can be set to ENABLE or DISABLE.

#### 9.1.2 CAN\_FilterTypeDef

*CAN\_FilterTypeDef* is defined in the `stm32f4xx_hal_can.h`

**Data Fields**

- *uint32\_t FilterIdHigh*
- *uint32\_t FilterIdLow*
- *uint32\_t FilterMaskIdHigh*
- *uint32\_t FilterMaskIdLow*
- *uint32\_t FilterFIFOAssignment*
- *uint32\_t FilterBank*
- *uint32\_t FilterMode*
- *uint32\_t FilterScale*
- *uint32\_t FilterActivation*
- *uint32\_t SlaveStartFilterBank*

**Field Documentation**

- ***uint32\_t CAN\_FilterTypeDef::FilterIdHigh***  
Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFF`.
- ***uint32\_t CAN\_FilterTypeDef::FilterIdLow***  
Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFF`.
- ***uint32\_t CAN\_FilterTypeDef::FilterMaskIdHigh***  
Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFF`.
- ***uint32\_t CAN\_FilterTypeDef::FilterMaskIdLow***  
Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFF`.
- ***uint32\_t CAN\_FilterTypeDef::FilterFIFOAssignment***  
Specifies the FIFO (0 or 1U) which will be assigned to the filter. This parameter can be a value of [CAN\\_filter\\_FIFO](#)
- ***uint32\_t CAN\_FilterTypeDef::FilterBank***  
Specifies the filter bank which will be initialized. For single CAN instance(14 dedicated filter banks), this parameter must be a number between `Min_Data = 0` and `Max_Data = 13`. For dual CAN instances(28 filter banks shared), this parameter must be a number between `Min_Data = 0` and `Max_Data = 27`.
- ***uint32\_t CAN\_FilterTypeDef::FilterMode***  
Specifies the filter mode to be initialized. This parameter can be a value of [CAN\\_filter\\_mode](#)
- ***uint32\_t CAN\_FilterTypeDef::FilterScale***  
Specifies the filter scale. This parameter can be a value of [CAN\\_filter\\_scale](#)
- ***uint32\_t CAN\_FilterTypeDef::FilterActivation***  
Enable or disable the filter. This parameter can be a value of [CAN\\_filter\\_activation](#)
- ***uint32\_t CAN\_FilterTypeDef::SlaveStartFilterBank***  
Select the start filter bank for the slave CAN instance. For single CAN instances, this parameter is meaningless. For dual CAN instances, all filter banks with lower index are assigned to master CAN instance, whereas all filter banks with greater index are assigned to slave CAN instance. This parameter must be a number between `Min_Data = 0` and `Max_Data = 27`.

**9.1.3**
**CAN\_TxHeaderTypeDef**

**CAN\_TxHeaderTypeDef** is defined in the `stm32f4xx_hal_can.h`

**Data Fields**

- *uint32\_t StdId*
- *uint32\_t ExtId*
- *uint32\_t IDE*
- *uint32\_t RTR*



- *uint32\_t* **DLC**
- *FunctionalState* **TransmitGlobalTime**

#### Field Documentation

- *uint32\_t* **CAN\_TxHeaderTypeDef::StdId**  
Specifies the standard identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x7FF.
- *uint32\_t* **CAN\_TxHeaderTypeDef::ExtId**  
Specifies the extended identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x1FFFFFFF.
- *uint32\_t* **CAN\_TxHeaderTypeDef::IDE**  
Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of [CAN\\_identifier\\_type](#)
- *uint32\_t* **CAN\_TxHeaderTypeDef::RTR**  
Specifies the type of frame for the message that will be transmitted. This parameter can be a value of [CAN\\_remote\\_transmission\\_request](#)
- *uint32\_t* **CAN\_TxHeaderTypeDef::DLC**  
Specifies the length of the frame that will be transmitted. This parameter must be a number between Min\_Data = 0 and Max\_Data = 8.
- *FunctionalState* **CAN\_TxHeaderTypeDef::TransmitGlobalTime**  
Specifies whether the timestamp counter value captured on start of frame transmission, is sent in DATA6 and DATA7 replacing pData[6] and pData[7].  
**Note:**
  - : Time Triggered Communication Mode must be enabled.
  - : DLC must be programmed as 8 bytes, in order these 2 bytes are sent. This parameter can be set to ENABLE or DISABLE.

### 9.1.4

#### CAN\_RxHeaderTypeDef

*CAN\_RxHeaderTypeDef* is defined in the stm32f4xx\_hal\_can.h

#### Data Fields

- *uint32\_t* **StdId**
- *uint32\_t* **ExtId**
- *uint32\_t* **IDE**
- *uint32\_t* **RTR**
- *uint32\_t* **DLC**
- *uint32\_t* **Timestamp**
- *uint32\_t* **FilterMatchIndex**

#### Field Documentation

- *uint32\_t* **CAN\_RxHeaderTypeDef::StdId**  
Specifies the standard identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x7FF.
- *uint32\_t* **CAN\_RxHeaderTypeDef::ExtId**  
Specifies the extended identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x1FFFFFFF.
- *uint32\_t* **CAN\_RxHeaderTypeDef::IDE**  
Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of [CAN\\_identifier\\_type](#)
- *uint32\_t* **CAN\_RxHeaderTypeDef::RTR**  
Specifies the type of frame for the message that will be transmitted. This parameter can be a value of [CAN\\_remote\\_transmission\\_request](#)
- *uint32\_t* **CAN\_RxHeaderTypeDef::DLC**  
Specifies the length of the frame that will be transmitted. This parameter must be a number between Min\_Data = 0 and Max\_Data = 8.

- ***uint32\_t CAN\_RxHeaderTypeDef::Timestamp***  
Specifies the timestamp counter value captured on start of frame reception.  
**Note:**
  - : Time Triggered Communication Mode must be enabled. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFFFF.
- ***uint32\_t CAN\_RxHeaderTypeDef::FilterMatchIndex***  
Specifies the index of matching acceptance filter element. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFF.

### 9.1.5 **\_\_CAN\_HandleTypeDef**

**\_\_CAN\_HandleTypeDef** is defined in the `stm32f4xx_hal_can.h`

#### Data Fields

- ***CAN\_TypeDef \* Instance***
- ***CAN\_InitTypeDef Init***
- ***\_\_IO HAL\_CAN\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***CAN\_TypeDef\* \_\_CAN\_HandleTypeDef::Instance***  
Register base address
- ***CAN\_InitTypeDef \_\_CAN\_HandleTypeDef::Init***  
CAN required parameters
- ***\_\_IO HAL\_CAN\_StateTypeDef \_\_CAN\_HandleTypeDef::State***  
CAN communication state
- ***\_\_IO uint32\_t \_\_CAN\_HandleTypeDef::ErrorCode***  
CAN Error code. This parameter can be a value of [CAN\\_Error\\_Code](#)

## 9.2 CAN Firmware driver API description

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

### 9.2.1 How to use this driver

1. Initialize the CAN low level resources by implementing the `HAL_CAN_MspInit()`:
  - Enable the CAN interface clock using `__HAL_RCC_CANx_CLK_ENABLE()`
  - Configure CAN pins
    - Enable the clock for the CAN GPIOs
    - Configure CAN pins as alternate function open-drain
  - In case of using interrupts (e.g. `HAL_CAN_ActivateNotification()`)
    - Configure the CAN interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the CAN IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In CAN IRQ handler, call `HAL_CAN_IRQHandler()`
2. Initialize the CAN peripheral using `HAL_CAN_Init()` function. This function resorts to `HAL_CAN_MspInit()` for low-level initialization.
3. Configure the reception filters using the following configuration functions:
  - `HAL_CAN_ConfigFilter()`
4. Start the CAN module using `HAL_CAN_Start()` function. At this level the node is active on the bus: it receive messages, and can send messages.

5. To manage messages transmission, the following Tx control functions can be used:
  - HAL\_CAN\_AddTxMessage() to request transmission of a new message.
  - HAL\_CAN\_AbortTxRequest() to abort transmission of a pending message.
  - HAL\_CAN\_GetTxMailboxesFreeLevel() to get the number of free Tx mailboxes.
  - HAL\_CAN\_IsTxMessagePending() to check if a message is pending in a Tx mailbox.
  - HAL\_CAN\_GetTxTimestamp() to get the timestamp of Tx message sent, if time triggered communication mode is enabled.
6. When a message is received into the CAN Rx FIFOs, it can be retrieved using the HAL\_CAN\_GetRxMessage() function. The function HAL\_CAN\_GetRxFifoFillLevel() allows to know how many Rx message are stored in the Rx Fifo.
7. Calling the HAL\_CAN\_Stop() function stops the CAN module.
8. The deinitialization is achieved with HAL\_CAN\_DeInit() function.

### Polling mode operation

1. Reception:
  - Monitor reception of message using HAL\_CAN\_GetRxFifoFillLevel() until at least one message is received.
  - Then get the message using HAL\_CAN\_GetRxMessage().
2. Transmission:
  - Monitor the Tx mailboxes availability until at least one Tx mailbox is free, using HAL\_CAN\_GetTxMailboxesFreeLevel().
  - Then request transmission of a message using HAL\_CAN\_AddTxMessage().

### Interrupt mode operation

1. Notifications are activated using HAL\_CAN\_ActivateNotification() function. Then, the process can be controlled through the available user callbacks: HAL\_CAN\_xxxCallback(), using same APIs HAL\_CAN\_GetRxMessage() and HAL\_CAN\_AddTxMessage().
2. Notifications can be deactivated using HAL\_CAN\_DeactivateNotification() function.
3. Special care should be taken for CAN\_IT\_RX\_FIFO0\_MSG\_PENDING and CAN\_IT\_RX\_FIFO1\_MSG\_PENDING notifications. These notifications trig the callbacks HAL\_CAN\_RxFIFO0MsgPendingCallback() and HAL\_CAN\_RxFIFO1MsgPendingCallback(). User has two possible options here.
  - Directly get the Rx message in the callback, using HAL\_CAN\_GetRxMessage().
  - Or deactivate the notification in the callback without getting the Rx message. The Rx message can then be got later using HAL\_CAN\_GetRxMessage(). Once the Rx message have been read, the notification can be activated again.

### Sleep mode

1. The CAN peripheral can be put in sleep mode (low power), using HAL\_CAN\_RequestSleep(). The sleep mode will be entered as soon as the current CAN activity (transmission or reception of a CAN frame) will be completed.
2. A notification can be activated to be informed when the sleep mode will be entered.
3. It can be checked if the sleep mode is entered using HAL\_CAN\_IsSleepActive(). Note that the CAN state (accessible from the API HAL\_CAN\_GetState()) is HAL\_CAN\_STATE\_SLEEP\_PENDING as soon as the sleep mode request is submitted (the sleep mode is not yet entered), and become HAL\_CAN\_STATE\_SLEEP\_ACTIVE when the sleep mode is effective.
4. The wake-up from sleep mode can be triggered by two ways:
  - Using HAL\_CAN\_WakeUp(). When returning from this function, the sleep mode is exited (if return status is HAL\_OK).
  - When a start of Rx CAN frame is detected by the CAN peripheral, if automatic wake up mode is enabled.

### Callback registration

### 9.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- HAL\_CAN\_Init : Initialize and configure the CAN.
- HAL\_CAN\_DeInit : De-initialize the CAN.
- HAL\_CAN\_MspInit : Initialize the CAN MSP.
- HAL\_CAN\_MspDeInit : DeInitialize the CAN MSP.

This section contains the following APIs:

- *HAL\_CAN\_Init()*
- *HAL\_CAN\_DeInit()*
- *HAL\_CAN\_MspInit()*
- *HAL\_CAN\_MspDeInit()*

### 9.2.3 Configuration functions

This section provides functions allowing to:

- HAL\_CAN\_ConfigFilter : Configure the CAN reception filters

This section contains the following APIs:

- *HAL\_CAN\_ConfigFilter()*

### 9.2.4 Control functions

This section provides functions allowing to:

- HAL\_CAN\_Start : Start the CAN module
- HAL\_CAN\_Stop : Stop the CAN module
- HAL\_CAN\_RequestSleep : Request sleep mode entry.
- HAL\_CAN\_WakeUp : Wake up from sleep mode.
- HAL\_CAN\_IsSleepActive : Check is sleep mode is active.
- HAL\_CAN\_AddTxMessage : Add a message to the Tx mailboxes and activate the corresponding transmission request
- HAL\_CAN\_AbortTxRequest : Abort transmission request
- HAL\_CAN\_GetTxMailboxesFreeLevel : Return Tx mailboxes free level
- HAL\_CAN\_IsTxMessagePending : Check if a transmission request is pending on the selected Tx mailbox
- HAL\_CAN\_GetRxMessage : Get a CAN frame from the Rx FIFO
- HAL\_CAN\_GetRxFifoFillLevel : Return Rx FIFO fill level

This section contains the following APIs:

- *HAL\_CAN\_Start()*
- *HAL\_CAN\_Stop()*
- *HAL\_CAN\_RequestSleep()*
- *HAL\_CAN\_WakeUp()*
- *HAL\_CAN\_IsSleepActive()*
- *HAL\_CAN\_AddTxMessage()*
- *HAL\_CAN\_AbortTxRequest()*
- *HAL\_CAN\_GetTxMailboxesFreeLevel()*
- *HAL\_CAN\_IsTxMessagePending()*
- *HAL\_CAN\_GetTxTimestamp()*
- *HAL\_CAN\_GetRxMessage()*
- *HAL\_CAN\_GetRxFifoFillLevel()*

### 9.2.5 Interrupts management

This section provides functions allowing to:

- HAL\_CAN\_ActivateNotification : Enable interrupts

- HAL\_CAN\_DeactivateNotification : Disable interrupts
- HAL\_CAN\_IRQHandler : Handles CAN interrupt request

This section contains the following APIs:

- [HAL\\_CAN\\_ActivateNotification\(\)](#)
- [HAL\\_CAN\\_DeactivateNotification\(\)](#)
- [HAL\\_CAN\\_IRQHandler\(\)](#)

### 9.2.6 Peripheral State and Error functions

This subsection provides functions allowing to :

- HAL\_CAN\_GetState() : Return the CAN state.
- HAL\_CAN\_GetError() : Return the CAN error codes if any.
- HAL\_CAN\_ResetError(): Reset the CAN error codes if any.

This section contains the following APIs:

- [HAL\\_CAN\\_GetState\(\)](#)
- [HAL\\_CAN\\_GetError\(\)](#)
- [HAL\\_CAN\\_ResetError\(\)](#)

### 9.2.7 Detailed description of functions

#### HAL\_CAN\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_CAN\_Init (CAN\_HandleTypeDef \* hcan)**

##### Function description

Initializes the CAN peripheral according to the specified parameters in the CAN\_InitStruct.

##### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

##### Return values

- **HAL**: status

#### HAL\_CAN\_DeInit

##### Function name

**HAL\_StatusTypeDef HAL\_CAN\_DeInit (CAN\_HandleTypeDef \* hcan)**

##### Function description

Deinitializes the CAN peripheral registers to their default reset values.

##### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

##### Return values

- **HAL**: status

#### HAL\_CAN\_MspInit

##### Function name

**void HAL\_CAN\_MspInit (CAN\_HandleTypeDef \* hcan)**

### Function description

Initializes the CAN MSP.

### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **None**:

**HAL\_CAN\_MspDeInit**

### Function name

**void HAL\_CAN\_MspDeInit (CAN\_HandleTypeDef \* hcan)**

### Function description

Deinitializes the CAN MSP.

### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **None**:

**HAL\_CAN\_ConfigFilter**

### Function name

**HAL\_StatusTypeDef HAL\_CAN\_ConfigFilter (CAN\_HandleTypeDef \* hcan, CAN\_FilterTypeDef \* sFilterConfig)**

### Function description

Configures the CAN reception filter according to the specified parameters in the CAN\_FilterInitStruct.

### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **sFilterConfig**: pointer to a CAN\_FilterTypeDef structure that contains the filter configuration information.

### Return values

- **None**:

**HAL\_CAN\_Start**

### Function name

**HAL\_StatusTypeDef HAL\_CAN\_Start (CAN\_HandleTypeDef \* hcan)**

### Function description

Start the CAN module.

### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **HAL**: status

### HAL\_CAN\_Stop

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_Stop (CAN\_HandleTypeDef \* hcan)**

#### Function description

Stop the CAN module and enable access to configuration registers.

#### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

#### Return values

- **HAL**: status

### HAL\_CAN\_RequestSleep

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_RequestSleep (CAN\_HandleTypeDef \* hcan)**

#### Function description

Request the sleep mode (low power) entry.

#### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

#### Return values

- **HAL**: status.

### HAL\_CAN\_WakeUp

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_WakeUp (CAN\_HandleTypeDef \* hcan)**

#### Function description

Wake up from sleep mode.

#### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

#### Return values

- **HAL**: status.

### HAL\_CAN\_IsSleepActive

#### Function name

**uint32\_t HAL\_CAN\_IsSleepActive (CAN\_HandleTypeDef \* hcan)**

#### Function description

Check is sleep mode is active.

#### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **Status:**
  - 0 : Sleep mode is not active.
  - 1 : Sleep mode is active.

### HAL\_CAN\_AddTxMessage

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_AddTxMessage (CAN\_HandleTypeDef \* hcan, CAN\_TxHeaderTypeDef \* pHeader, uint8\_t aData, uint32\_t \* pTxMailbox)**

#### Function description

Add a message to the first free Tx mailbox and activate the corresponding transmission request.

#### Parameters

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **pHeader:** pointer to a CAN\_TxHeaderTypeDef structure.
- **aData:** array containing the payload of the Tx frame.
- **pTxMailbox:** pointer to a variable where the function will return the TxMailbox used to store the Tx message. This parameter can be a value of
  - CAN\_Tx\_Mailboxes.

### Return values

- **HAL:** status

### HAL\_CAN\_AbortTxRequest

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_AbortTxRequest (CAN\_HandleTypeDef \* hcan, uint32\_t TxMailboxes)**

#### Function description

Abort transmission requests.

#### Parameters

- **hcan:** pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **TxMailboxes:** List of the Tx Mailboxes to abort. This parameter can be any combination of
  - CAN\_Tx\_Mailboxes.

### Return values

- **HAL:** status

### HAL\_CAN\_GetTxMailboxesFreeLevel

#### Function name

**uint32\_t HAL\_CAN\_GetTxMailboxesFreeLevel (CAN\_HandleTypeDef \* hcan)**

#### Function description

Return Tx Mailboxes free level: number of free Tx Mailboxes.

#### Parameters

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **Number:** of free Tx Mailboxes.



## HAL\_CAN\_IsTxMessagePending

### Function name

**uint32\_t HAL\_CAN\_IsTxMessagePending (CAN\_HandleTypeDef \* hcan, uint32\_t TxMailboxes)**

### Function description

Check if a transmission request is pending on the selected Tx Mailboxes.

### Parameters

- **hcan:** pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **TxMailboxes:** List of Tx Mailboxes to check. This parameter can be any combination of
  - CAN\_Tx\_Mailboxes.

### Return values

- **Status:**
  - 0 : No pending transmission request on any selected Tx Mailboxes.
  - 1 : Pending transmission request on at least one of the selected Tx Mailbox.

## HAL\_CAN\_GetTxTimestamp

### Function name

**uint32\_t HAL\_CAN\_GetTxTimestamp (CAN\_HandleTypeDef \* hcan, uint32\_t TxMailbox)**

### Function description

Return timestamp of Tx message sent, if time triggered communication mode is enabled.

### Parameters

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **TxMailbox:** Tx Mailbox where the timestamp of message sent will be read. This parameter can be one value of
  - CAN\_Tx\_Mailboxes.

### Return values

- **Timestamp:** of message sent from Tx Mailbox.

## HAL\_CAN\_GetRxMessage

### Function name

**HAL\_StatusTypeDef HAL\_CAN\_GetRxMessage (CAN\_HandleTypeDef \* hcan, uint32\_t RxFifo, CAN\_RxHeaderTypeDef \* pHeader, uint8\_t aData)**

### Function description

Get an CAN frame from the Rx FIFO zone into the message RAM.

### Parameters

- **hcan:** pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **RxFifo:** Fifo number of the received message to be read. This parameter can be a value of
  - CAN\_receive\_FIFO\_number.
- **pHeader:** pointer to a CAN\_RxHeaderTypeDef structure where the header of the Rx frame will be stored.
- **aData:** array where the payload of the Rx frame will be stored.

### Return values

- **HAL:** status

### HAL\_CAN\_GetRxFifoFillLevel

#### Function name

**uint32\_t HAL\_CAN\_GetRxFifoFillLevel (CAN\_HandleTypeDef \* hcan, uint32\_t RxFifo)**

#### Function description

Return Rx FIFO fill level.

#### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **RxFifo**: Rx FIFO. This parameter can be a value of
  - CAN\_receive\_FIFO\_number.

#### Return values

- **Number**: of messages available in Rx FIFO.

### HAL\_CAN\_ActivateNotification

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_ActivateNotification (CAN\_HandleTypeDef \* hcan, uint32\_t ActiveITs)**

#### Function description

Enable interrupts.

#### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **ActiveITs**: indicates which interrupts will be enabled. This parameter can be any combination of
  - CAN\_Interrupts.

#### Return values

- **HAL**: status

### HAL\_CAN\_DeactivateNotification

#### Function name

**HAL\_StatusTypeDef HAL\_CAN\_DeactivateNotification (CAN\_HandleTypeDef \* hcan, uint32\_t InactiveITs)**

#### Function description

Disable interrupts.

#### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **InactiveITs**: indicates which interrupts will be disabled. This parameter can be any combination of
  - CAN\_Interrupts.

#### Return values

- **HAL**: status

### HAL\_CAN\_IRQHandler

#### Function name

**void HAL\_CAN\_IRQHandler (CAN\_HandleTypeDef \* hcan)**

**Function description**

Handles CAN interrupt request.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None**:

**HAL\_CAN\_TxMailbox0CompleteCallback**

**Function name**

**void HAL\_CAN\_TxMailbox0CompleteCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Transmission Mailbox 0 complete callback.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None**:

**HAL\_CAN\_TxMailbox1CompleteCallback**

**Function name**

**void HAL\_CAN\_TxMailbox1CompleteCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Transmission Mailbox 1 complete callback.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None**:

**HAL\_CAN\_TxMailbox2CompleteCallback**

**Function name**

**void HAL\_CAN\_TxMailbox2CompleteCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Transmission Mailbox 2 complete callback.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None**:

**HAL\_CAN\_TxMailbox0AbortCallback**

**Function name**

**void HAL\_CAN\_TxMailbox0AbortCallback (CAN\_HandleTypeDef \* hcan)**

### Function description

Transmission Mailbox 0 Cancellation callback.

### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **None**:

**HAL\_CAN\_TxMailbox1AbortCallback**

### Function name

**void HAL\_CAN\_TxMailbox1AbortCallback (CAN\_HandleTypeDef \* hcan)**

### Function description

Transmission Mailbox 1 Cancellation callback.

### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **None**:

**HAL\_CAN\_TxMailbox2AbortCallback**

### Function name

**void HAL\_CAN\_TxMailbox2AbortCallback (CAN\_HandleTypeDef \* hcan)**

### Function description

Transmission Mailbox 2 Cancellation callback.

### Parameters

- **hcan**: pointer to an CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **None**:

**HAL\_CAN\_RxFifo0MsgPendingCallback**

### Function name

**void HAL\_CAN\_RxFifo0MsgPendingCallback (CAN\_HandleTypeDef \* hcan)**

### Function description

Rx FIFO 0 message pending callback.

### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **None**:

**HAL\_CAN\_RxFifo0FullCallback**

### Function name

**void HAL\_CAN\_RxFifo0FullCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Rx FIFO 0 full callback.

**Parameters**

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None:**

**HAL\_CAN\_RxFifo1MsgPendingCallback**

**Function name**

**void HAL\_CAN\_RxFifo1MsgPendingCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Rx FIFO 1 message pending callback.

**Parameters**

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None:**

**HAL\_CAN\_RxFifo1FullCallback**

**Function name**

**void HAL\_CAN\_RxFifo1FullCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Rx FIFO 1 full callback.

**Parameters**

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None:**

**HAL\_CAN\_SleepCallback**

**Function name**

**void HAL\_CAN\_SleepCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Sleep callback.

**Parameters**

- **hcan:** pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None:**

**HAL\_CAN\_WakeUpFromRxMsgCallback**

**Function name**

**void HAL\_CAN\_WakeUpFromRxMsgCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

WakeUp from Rx message callback.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None**:

**HAL\_CAN\_ErrorCallback**

**Function name**

**void HAL\_CAN\_ErrorCallback (CAN\_HandleTypeDef \* hcan)**

**Function description**

Error CAN callback.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **None**:

**HAL\_CAN\_GetState**

**Function name**

**HAL\_CAN\_StateTypeDef HAL\_CAN\_GetState (CAN\_HandleTypeDef \* hcan)**

**Function description**

Return the CAN state.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **HAL**: state

**HAL\_CAN\_GetError**

**Function name**

**uint32\_t HAL\_CAN\_GetError (CAN\_HandleTypeDef \* hcan)**

**Function description**

Return the CAN error code.

**Parameters**

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

**Return values**

- **CAN**: Error Code

**HAL\_CAN\_ResetError**

**Function name**

**HAL\_StatusTypeDef HAL\_CAN\_ResetError (CAN\_HandleTypeDef \* hcan)**

### Function description

Reset the CAN error code.

### Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

### Return values

- **HAL**: status

## 9.3 CAN Firmware driver defines

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

### 9.3.1 CAN

CAN

*CAN Error Code*

#### HAL\_CAN\_ERROR\_NONE

No error

#### HAL\_CAN\_ERROR\_EWG

Protocol Error Warning

#### HAL\_CAN\_ERROR\_EPV

Error Passive

#### HAL\_CAN\_ERROR\_BOF

Bus-off error

#### HAL\_CAN\_ERROR\_STF

Stuff error

#### HAL\_CAN\_ERROR\_FOR

Form error

#### HAL\_CAN\_ERROR\_ACK

Acknowledgment error

#### HAL\_CAN\_ERROR\_BR

Bit recessive error

#### HAL\_CAN\_ERROR\_BD

Bit dominant error

#### HAL\_CAN\_ERROR\_CRC

CRC error

#### HAL\_CAN\_ERROR\_RX\_FOV0

Rx FIFO0 overrun error

#### HAL\_CAN\_ERROR\_RX\_FOV1

Rx FIFO1 overrun error

#### HAL\_CAN\_ERROR\_TX\_ALST0

TxMailbox 0 transmit failure due to arbitration lost

**HAL\_CAN\_ERROR\_TX\_TERR0**

TxMailbox 1 transmit failure due to transmit error

**HAL\_CAN\_ERROR\_TX\_ALST1**

TxMailbox 0 transmit failure due to arbitration lost

**HAL\_CAN\_ERROR\_TX\_TERR1**

TxMailbox 1 transmit failure due to transmit error

**HAL\_CAN\_ERROR\_TX\_ALST2**

TxMailbox 0 transmit failure due to arbitration lost

**HAL\_CAN\_ERROR\_TX\_TERR2**

TxMailbox 1 transmit failure due to transmit error

**HAL\_CAN\_ERROR\_TIMEOUT**

Timeout error

**HAL\_CAN\_ERROR\_NOT\_INITIALIZED**

Peripheral not initialized

**HAL\_CAN\_ERROR\_NOT\_READY**

Peripheral not ready

**HAL\_CAN\_ERROR\_NOT\_STARTED**

Peripheral not started

**HAL\_CAN\_ERROR\_PARAM**

Parameter error

**HAL\_CAN\_ERROR\_INTERNAL**

Internal error

***CAN Exported Macros***

**\_\_HAL\_CAN\_RESET\_HANDLE\_STATE**

**Description:**

- Reset CAN handle state.

**Parameters:**

- `__HANDLE__`: CAN handle.

**Return value:**

- None

**\_\_HAL\_CAN\_ENABLE\_IT**

**Description:**

- Enable the specified CAN interrupts.

**Parameters:**

- `__HANDLE__`: CAN handle.
- `__INTERRUPT__`: CAN Interrupt sources to enable. This parameter can be any combination of
  - `CAN_Interrupts`

**Return value:**

- None



### \_\_HAL\_CAN\_DISABLE\_IT

**Description:**

- Disable the specified CAN interrupts.

**Parameters:**

- `__HANDLE__`: CAN handle.
- `__INTERRUPT__`: CAN Interrupt sources to disable. This parameter can be any combination of
  - `CAN_Interrupts`

**Return value:**

- None

### \_\_HAL\_CAN\_GET\_IT\_SOURCE

**Description:**

- Check if the specified CAN interrupt source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: specifies the CAN Handle.
- `__INTERRUPT__`: specifies the CAN interrupt source to check. This parameter can be a value of
  - `CAN_Interrupts`

**Return value:**

- The: state of `__IT__` (TRUE or FALSE).

### \_\_HAL\_CAN\_GET\_FLAG

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the CAN Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of
  - `CAN_flags`

**Return value:**

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

## **\_\_HAL\_CAN\_CLEAR\_FLAG**

### **Description:**

- Clear the specified CAN pending flag.

### **Parameters:**

- **\_\_HANDLE\_\_**: specifies the CAN Handle.
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - CAN\_FLAG\_RQCP0: Request complete MailBox 0 Flag
  - CAN\_FLAG\_TXOK0: Transmission OK MailBox 0 Flag
  - CAN\_FLAG\_ALST0: Arbitration Lost MailBox 0 Flag
  - CAN\_FLAG\_TERR0: Transmission error MailBox 0 Flag
  - CAN\_FLAG\_RQCP1: Request complete MailBox 1 Flag
  - CAN\_FLAG\_TXOK1: Transmission OK MailBox 1 Flag
  - CAN\_FLAG\_ALST1: Arbitration Lost MailBox 1 Flag
  - CAN\_FLAG\_TERR1: Transmission error MailBox 1 Flag
  - CAN\_FLAG\_RQCP2: Request complete MailBox 2 Flag
  - CAN\_FLAG\_TXOK2: Transmission OK MailBox 2 Flag
  - CAN\_FLAG\_ALST2: Arbitration Lost MailBox 2 Flag
  - CAN\_FLAG\_TERR2: Transmission error MailBox 2 Flag
  - CAN\_FLAG\_FF0: RX FIFO 0 Full Flag
  - CAN\_FLAG\_FOV0: RX FIFO 0 Overrun Flag
  - CAN\_FLAG\_FF1: RX FIFO 1 Full Flag
  - CAN\_FLAG\_FOV1: RX FIFO 1 Overrun Flag
  - CAN\_FLAG\_WKUI: Wake up Interrupt Flag
  - CAN\_FLAG\_SLAKI: Sleep acknowledge Interrupt Flag

### **Return value:**

- None

### **CAN Filter Activation**

#### **CAN\_FILTER\_DISABLE**

Disable filter

#### **CAN\_FILTER\_ENABLE**

Enable filter

### **CAN Filter FIFO**

#### **CAN\_FILTER\_FIFO0**

Filter FIFO 0 assignment for filter x

#### **CAN\_FILTER\_FIFO1**

Filter FIFO 1 assignment for filter x

### **CAN Filter Mode**

#### **CAN\_FILTERMODE\_IDMASK**

Identifier mask mode

#### **CAN\_FILTERMODE\_IDLIST**

Identifier list mode

### **CAN Filter Scale**

#### **CAN\_FILTERSCALE\_16BIT**

Two 16-bit filters

**CAN\_FILTERSCALE\_32BIT**

One 32-bit filter

**CAN Flags****CAN\_FLAG\_RQCP0**

Request complete MailBox 0 flag

**CAN\_FLAG\_TXOK0**

Transmission OK MailBox 0 flag

**CAN\_FLAG\_ALST0**

Arbitration Lost MailBox 0 flag

**CAN\_FLAG\_TERR0**

Transmission error MailBox 0 flag

**CAN\_FLAG\_RQCP1**

Request complete MailBox1 flag

**CAN\_FLAG\_TXOK1**

Transmission OK MailBox 1 flag

**CAN\_FLAG\_ALST1**

Arbitration Lost MailBox 1 flag

**CAN\_FLAG\_TERR1**

Transmission error MailBox 1 flag

**CAN\_FLAG\_RQCP2**

Request complete MailBox2 flag

**CAN\_FLAG\_TXOK2**

Transmission OK MailBox 2 flag

**CAN\_FLAG\_ALST2**

Arbitration Lost MailBox 2 flag

**CAN\_FLAG\_TERR2**

Transmission error MailBox 2 flag

**CAN\_FLAG\_TME0**

Transmit mailbox 0 empty flag

**CAN\_FLAG\_TME1**

Transmit mailbox 1 empty flag

**CAN\_FLAG\_TME2**

Transmit mailbox 2 empty flag

**CAN\_FLAG\_LOW0**

Lowest priority mailbox 0 flag

**CAN\_FLAG\_LOW1**

Lowest priority mailbox 1 flag

**CAN\_FLAG\_LOW2**

Lowest priority mailbox 2 flag

**CAN\_FLAG\_FF0**

RX FIFO 0 Full flag

**CAN\_FLAG\_FOV0**

RX FIFO 0 Overrun flag

**CAN\_FLAG\_FF1**

RX FIFO 1 Full flag

**CAN\_FLAG\_FOV1**

RX FIFO 1 Overrun flag

**CAN\_FLAG\_INAK**

Initialization acknowledge flag

**CAN\_FLAG\_SLAK**

Sleep acknowledge flag

**CAN\_FLAG\_ERRI**

Error flag

**CAN\_FLAG\_WKU**

Wake up interrupt flag

**CAN\_FLAG\_SLAKI**

Sleep acknowledge interrupt flag

**CAN\_FLAG\_EWG**

Error warning flag

**CAN\_FLAG\_EPV**

Error passive flag

**CAN\_FLAG\_BOF**

Bus-Off flag

**CAN Identifier Type****CAN\_ID\_STD**

Standard Id

**CAN\_ID\_EXT**

Extended Id

**CAN InitStatus****CAN\_INITSTATUS\_FAILED**

CAN initialization failed

**CAN\_INITSTATUS\_SUCCESS**

CAN initialization OK

**CAN Interrupts****CAN\_IT\_TX\_MAILBOX\_EMPTY**

Transmit mailbox empty interrupt

**CAN\_IT\_RX\_FIFO0\_MSG\_PENDING**

FIFO 0 message pending interrupt

**CAN\_IT\_RX\_FIFO0\_FULL**

FIFO 0 full interrupt

**CAN\_IT\_RX\_FIFO0\_OVERRUN**

FIFO 0 overrun interrupt

**CAN\_IT\_RX\_FIFO1\_MSG\_PENDING**

FIFO 1 message pending interrupt

**CAN\_IT\_RX\_FIFO1\_FULL**

FIFO 1 full interrupt

**CAN\_IT\_RX\_FIFO1\_OVERRUN**

FIFO 1 overrun interrupt

**CAN\_IT\_WAKEUP**

Wake-up interrupt

**CAN\_IT\_SLEEP\_ACK**

Sleep acknowledge interrupt

**CAN\_IT\_ERROR\_WARNING**

Error warning interrupt

**CAN\_IT\_ERROR\_PASSIVE**

Error passive interrupt

**CAN\_IT\_BUSOFF**

Bus-off interrupt

**CAN\_IT\_LAST\_ERROR\_CODE**

Last error code interrupt

**CAN\_IT\_ERROR**

Error Interrupt

***CAN Operating Mode*****CAN\_MODE\_NORMAL**

Normal mode

**CAN\_MODE\_LOOPBACK**

Loopback mode

**CAN\_MODE\_SILENT**

Silent mode

**CAN\_MODE\_SILENT\_LOOPBACK**

Loopback combined with silent mode

***CAN Receive FIFO Number*****CAN\_RX\_FIFO0**

CAN receive FIFO 0

**CAN\_RX\_FIFO1**

CAN receive FIFO 1

***CAN Remote Transmission Request***

**CAN\_RTR\_DATA**

Data frame

**CAN\_RTR\_REMOTE**

Remote frame

***CAN Synchronization Jump Width*****CAN\_SJW\_1TQ**

1 time quantum

**CAN\_SJW\_2TQ**

2 time quantum

**CAN\_SJW\_3TQ**

3 time quantum

**CAN\_SJW\_4TQ**

4 time quantum

***CAN Time Quantum in Bit Segment 1*****CAN\_BS1\_1TQ**

1 time quantum

**CAN\_BS1\_2TQ**

2 time quantum

**CAN\_BS1\_3TQ**

3 time quantum

**CAN\_BS1\_4TQ**

4 time quantum

**CAN\_BS1\_5TQ**

5 time quantum

**CAN\_BS1\_6TQ**

6 time quantum

**CAN\_BS1\_7TQ**

7 time quantum

**CAN\_BS1\_8TQ**

8 time quantum

**CAN\_BS1\_9TQ**

9 time quantum

**CAN\_BS1\_10TQ**

10 time quantum

**CAN\_BS1\_11TQ**

11 time quantum

**CAN\_BS1\_12TQ**

12 time quantum

**CAN\_BS1\_13TQ**

13 time quantum

**CAN\_BS1\_14TQ**

14 time quantum

**CAN\_BS1\_15TQ**

15 time quantum

**CAN\_BS1\_16TQ**

16 time quantum

***CAN Time Quantum in Bit Segment 2*****CAN\_BS2\_1TQ**

1 time quantum

**CAN\_BS2\_2TQ**

2 time quantum

**CAN\_BS2\_3TQ**

3 time quantum

**CAN\_BS2\_4TQ**

4 time quantum

**CAN\_BS2\_5TQ**

5 time quantum

**CAN\_BS2\_6TQ**

6 time quantum

**CAN\_BS2\_7TQ**

7 time quantum

**CAN\_BS2\_8TQ**

8 time quantum

***CAN Tx Mailboxes*****CAN\_TX\_MAILBOX0**

Tx Mailbox 0

**CAN\_TX\_MAILBOX1**

Tx Mailbox 1

**CAN\_TX\_MAILBOX2**

Tx Mailbox 2

## 10 HAL CEC Generic Driver

### 10.1 CEC Firmware driver registers structures

#### 10.1.1 CEC\_InitTypeDef

**CEC\_InitTypeDef** is defined in the `stm32f4xx_hal_cec.h`

##### Data Fields

- `uint32_t SignalFreeTime`
- `uint32_t Tolerance`
- `uint32_t BRERxStop`
- `uint32_t BREErrorBitGen`
- `uint32_t LBPEErrorBitGen`
- `uint32_t BroadcastMsgNoErrorBitGen`
- `uint32_t SignalFreeTimeOption`
- `uint32_t ListenMode`
- `uint16_t OwnAddress`
- `uint8_t * RxBuffer`

##### Field Documentation

- **`uint32_t CEC_InitTypeDef::SignalFreeTime`**  
Set SFT field, specifies the Signal Free Time. It can be one of **`CEC_Signal_Free_Time`** and belongs to the set {0,...,7} where 0x0 is the default configuration else means  $0.5 + (\text{SignalFreeTime} - 1)$  nominal data bit periods
- **`uint32_t CEC_InitTypeDef::Tolerance`**  
Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of **`CEC_Tolerance`**: it is either `CEC_STANDARD_TOLERANCE` or `CEC_EXTENDED_TOLERANCE`
- **`uint32_t CEC_InitTypeDef::BRERxStop`**  
Set BRESTP bit **`CEC_BRERxStop`**: specifies whether or not a Bit Rising Error stops the reception. `CEC_NO_RX_STOP_ON_BRE`: reception is not stopped. `CEC_RX_STOP_ON_BRE`: reception is stopped.
- **`uint32_t CEC_InitTypeDef::BREErrorBitGen`**  
Set BREGEN bit **`CEC_BREErrorBitGen`**: specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection. `CEC_BRE_ERRORBIT_NO_GENERATION`: no error-bit generation. `CEC_BRE_ERRORBIT_GENERATION`: error-bit generation if BRESTP is set.
- **`uint32_t CEC_InitTypeDef::LBPEErrorBitGen`**  
Set LBPEGEN bit **`CEC_LBPEErrorBitGen`**: specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection. `CEC_LBPE_ERRORBIT_NO_GENERATION`: no error-bit generation. `CEC_LBPE_ERRORBIT_GENERATION`: error-bit generation.
- **`uint32_t CEC_InitTypeDef::BroadcastMsgNoErrorBitGen`**  
Set BRDNOGEN bit **`CEC_BroadCastMsgErrorBitGen`**: allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message. It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values: 1) `CEC_BROADCASTERROR_ERRORBIT_GENERATION`. a) BRE detection: error-bit generation on the CEC line if `BRESTP=CEC_RX_STOP_ON_BRE` and `BREGEN=CEC_BRE_ERRORBIT_NO_GENERATION`. b) LBPE detection: error-bit generation on the CEC line if `LBPGEN=CEC_LBPE_ERRORBIT_NO_GENERATION`. 2) `CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION`. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.
- **`uint32_t CEC_InitTypeDef::SignalFreeTimeOption`**  
Set SFTOP bit **`CEC_SFT_Option`**: specifies when SFT timer starts. `CEC_SFT_START_ON_TXSOM` SFT: timer starts when TXSOM is set by software. `CEC_SFT_START_ON_TX_RX_END`: SFT timer starts automatically at the end of message transmission/reception.



- ***uint32\_t CEC\_InitTypeDef::ListenMode***  
Set LSTN bit ***CEC\_Listening\_Mode*** : specifies device listening mode. It can take two values: ***CEC\_REDUCED\_LISTENING\_MODE***: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received. ***CEC\_FULL\_LISTENING\_MODE***: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- ***uint16\_t CEC\_InitTypeDef::OwnAddress***  
Own addresses configuration This parameter can be a value of ***CEC\_OWN\_ADDRESS***
- ***uint8\_t\* CEC\_InitTypeDef::RxBuffer***  
CEC Rx buffer pointer

### 10.1.2

#### CEC\_HandleTypeDef

***CEC\_HandleTypeDef*** is defined in the `stm32f4xx_hal_cec.h`

##### Data Fields

- ***CEC\_TypeDef \* Instance***
- ***CEC\_InitTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferCount***
- ***uint16\_t RxXferSize***
- ***HAL\_LockTypeDef Lock***
- ***HAL\_CEC\_StateTypeDef gState***
- ***HAL\_CEC\_StateTypeDef RxState***
- ***uint32\_t ErrorCode***

##### Field Documentation

- ***CEC\_TypeDef\* CEC\_HandleTypeDef::Instance***  
CEC registers base address
- ***CEC\_InitTypeDef CEC\_HandleTypeDef::Init***  
CEC communication parameters
- ***uint8\_t\* CEC\_HandleTypeDef::pTxBuffPtr***  
Pointer to CEC Tx transfer Buffer
- ***uint16\_t CEC\_HandleTypeDef::TxXferCount***  
CEC Tx Transfer Counter
- ***uint16\_t CEC\_HandleTypeDef::RxXferSize***  
CEC Rx Transfer size, 0: header received only
- ***HAL\_LockTypeDef CEC\_HandleTypeDef::Lock***  
Locking object
- ***HAL\_CEC\_StateTypeDef CEC\_HandleTypeDef::gState***  
CEC state information related to global Handle management and also related to Tx operations. This parameter can be a value of ***HAL\_CEC\_StateTypeDef***
- ***HAL\_CEC\_StateTypeDef CEC\_HandleTypeDef::RxState***  
CEC state information related to Rx operations. This parameter can be a value of ***HAL\_CEC\_StateTypeDef***
- ***uint32\_t CEC\_HandleTypeDef::ErrorCode***  
For errors handling purposes, copy of ISR register in case error is reported

## 10.2

### CEC Firmware driver API description

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

#### 10.2.1

##### How to use this driver

The CEC HAL driver can be used as follow:

1. Declare a ***CEC\_HandleTypeDef*** handle structure.

2. Initialize the CEC low level resources by implementing the HAL\_CEC\_MspInit ()API:
  - a. Enable the CEC interface clock.
  - b. CEC pins configuration:
    - Enable the clock for the CEC GPIOs.
    - Configure these CEC pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_CEC\_Transmit\_IT() and HAL\_CEC\_Receive\_IT() APIs):
    - Configure the CEC interrupt priority.
    - Enable the NVIC CEC IRQ handle.
    - The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_HAL\_CEC\_ENABLE\_IT() and \_\_HAL\_CEC\_DISABLE\_IT() inside the transmit and receive process.
3. Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL\_CEC\_Init() API.

*Note:* This API (HAL\_CEC\_Init()) configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customed HAL\_CEC\_MspInit() API.

### Callback registration

## 10.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the CEC

- The following parameters need to be configured:
  - SignalFreeTime
  - Tolerance
  - BRERxStop (RX stopped or not upon Bit Rising Error)
  - BREErrorBitGen (Error-Bit generation in case of Bit Rising Error)
  - LBPEErrorBitGen (Error-Bit generation in case of Long Bit Period Error)
  - BroadcastMsgNoErrorBitGen (Error-bit generation in case of broadcast message error)
  - SignalFreeTimeOption (SFT Timer start definition)
  - OwnAddress (CEC device address)
  - ListenMode

This section contains the following APIs:

- [HAL\\_CEC\\_Init\(\)](#)
- [HAL\\_CEC\\_DeInit\(\)](#)
- [HAL\\_CEC\\_SetDeviceAddress\(\)](#)
- [HAL\\_CEC\\_MspInit\(\)](#)
- [HAL\\_CEC\\_MspDeInit\(\)](#)

## 10.2.3 IO operation functions

This section contains the following APIs:

- [HAL\\_CEC\\_Transmit\\_IT\(\)](#)
- [HAL\\_CEC\\_GetLastReceivedFrameSize\(\)](#)
- [HAL\\_CEC\\_ChangeRxBuffer\(\)](#)
- [HAL\\_CEC\\_IRQHandler\(\)](#)
- [HAL\\_CEC\\_TxCpltCallback\(\)](#)
- [HAL\\_CEC\\_RxCpltCallback\(\)](#)
- [HAL\\_CEC\\_ErrorCallback\(\)](#)

## 10.2.4 Peripheral Control function

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

- HAL\_CEC\_GetState() API can be helpful to check in run-time the state of the CEC peripheral.
- HAL\_CEC\_GetError() API can be helpful to check in run-time the error of the CEC peripheral.

This section contains the following APIs:

- [HAL\\_CEC\\_GetState\(\)](#)
- [HAL\\_CEC\\_GetError\(\)](#)

## 10.2.5 Detailed description of functions

### HAL\_CEC\_Init

#### Function name

**HAL\_StatusTypeDef HAL\_CEC\_Init (CEC\_HandleTypeDef \* hcec)**

#### Function description

Initializes the CEC mode according to the specified parameters in the CEC\_InitTypeDef and creates the associated handle .

#### Parameters

- **hcec**: CEC handle

#### Return values

- **HAL**: status

### HAL\_CEC\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_CEC\_DeInit (CEC\_HandleTypeDef \* hcec)**

#### Function description

DeInitializes the CEC peripheral.

#### Parameters

- **hcec**: CEC handle

#### Return values

- **HAL**: status

### HAL\_CEC\_SetDeviceAddress

#### Function name

**HAL\_StatusTypeDef HAL\_CEC\_SetDeviceAddress (CEC\_HandleTypeDef \* hcec, uint16\_t CEC\_OwnAddress)**

#### Function description

Initializes the Own Address of the CEC device.

#### Parameters

- **hcec**: CEC handle
- **CEC\_OwnAddress**: The CEC own address.

#### Return values

- **HAL**: status

### HAL\_CEC\_Msplnit

#### Function name

**void HAL\_CEC\_Msplnit (CEC\_HandleTypeDef \* hcec)**

### Function description

CEC MSP Init.

### Parameters

- **hcec**: CEC handle

### Return values

- **None**:

### HAL\_CEC\_MspDeInit

### Function name

**void HAL\_CEC\_MspDeInit (CEC\_HandleTypeDef \* hcec)**

### Function description

CEC MSP DeInit.

### Parameters

- **hcec**: CEC handle

### Return values

- **None**:

### HAL\_CEC\_Transmit\_IT

### Function name

**HAL\_StatusTypeDef HAL\_CEC\_Transmit\_IT (CEC\_HandleTypeDef \* hcec, uint8\_t InitiatorAddress, uint8\_t DestinationAddress, uint8\_t \* pData, uint32\_t Size)**

### Function description

Send data in interrupt mode.

### Parameters

- **hcec**: CEC handle
- **InitiatorAddress**: Initiator address
- **DestinationAddress**: destination logical address
- **pData**: pointer to input byte data buffer
- **Size**: amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).

### Return values

- **HAL**: status

### HAL\_CEC\_GetLastReceivedFrameSize

### Function name

**uint32\_t HAL\_CEC\_GetLastReceivedFrameSize (CEC\_HandleTypeDef \* hcec)**

### Function description

Get size of the received frame.

### Parameters

- **hcec**: CEC handle

### Return values

- **Frame**: size

### HAL\_CEC\_ChangeRxBuffer

#### Function name

**void HAL\_CEC\_ChangeRxBuffer (CEC\_HandleTypeDef \* hcec, uint8\_t \* Rxbuffer)**

#### Function description

Change Rx Buffer.

#### Parameters

- **hcec:** CEC handle
- **Rxbuffer:** Rx Buffer

#### Return values

- **Frame:** size

#### Notes

- This function can be called only inside the HAL\_CEC\_RxCpltCallback()

### HAL\_CEC\_IRQHandler

#### Function name

**void HAL\_CEC\_IRQHandler (CEC\_HandleTypeDef \* hcec)**

#### Function description

This function handles CEC interrupt requests.

#### Parameters

- **hcec:** CEC handle

#### Return values

- **None:**

### HAL\_CEC\_TxCpltCallback

#### Function name

**void HAL\_CEC\_TxCpltCallback (CEC\_HandleTypeDef \* hcec)**

#### Function description

Tx Transfer completed callback.

#### Parameters

- **hcec:** CEC handle

#### Return values

- **None:**

### HAL\_CEC\_RxCpltCallback

#### Function name

**void HAL\_CEC\_RxCpltCallback (CEC\_HandleTypeDef \* hcec, uint32\_t RxFrameSize)**

#### Function description

Rx Transfer completed callback.

#### Parameters

- **hcec:** CEC handle
- **RxFrameSize:** Size of frame

**Return values**

- **None:**

**HAL\_CEC\_ErrorCallback**

**Function name**

**void HAL\_CEC\_ErrorCallback (CEC\_HandleTypeDef \* hcec)**

**Function description**

CEC error callbacks.

**Parameters**

- **hcec:** CEC handle

**Return values**

- **None:**

**HAL\_CEC\_GetState**

**Function name**

**HAL\_CEC\_StateTypeDef HAL\_CEC\_GetState (CEC\_HandleTypeDef \* hcec)**

**Function description**

return the CEC state

**Parameters**

- **hcec:** pointer to a CEC\_HandleTypeDef structure that contains the configuration information for the specified CEC module.

**Return values**

- **HAL:** state

**HAL\_CEC\_GetError**

**Function name**

**uint32\_t HAL\_CEC\_GetError (CEC\_HandleTypeDef \* hcec)**

**Function description**

Return the CEC error code.

**Parameters**

- **hcec:** pointer to a CEC\_HandleTypeDef structure that contains the configuration information for the specified CEC.

**Return values**

- **CEC:** Error Code

## 10.3 CEC Firmware driver defines

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

### 10.3.1 CEC

CEC

*CEC all RX or TX errors flags*

**CEC\_ISR\_ALL\_ERROR**

*CEC Error Bit Generation if Bit Rise Error reported*

CEC\_BRE\_ERRORBIT\_NO\_GENERATION

CEC\_BRE\_ERRORBIT\_GENERATION

*CEC Reception Stop on Error*

CEC\_NO\_RX\_STOP\_ON\_BRE

CEC\_RX\_STOP\_ON\_BRE

*CEC Error Bit Generation on Broadcast message*

CEC\_BROADCASTERROR\_ERRORBIT\_GENERATION

CEC\_BROADCASTERROR\_NO\_ERRORBIT\_GENERATION

*CEC Error Code*

HAL\_CEC\_ERROR\_NONE

no error

HAL\_CEC\_ERROR\_RXOVR

CEC Rx-Overrun

HAL\_CEC\_ERROR\_BRE

CEC Rx Bit Rising Error

HAL\_CEC\_ERROR\_SBPE

CEC Rx Short Bit period Error

HAL\_CEC\_ERROR\_LBPE

CEC Rx Long Bit period Error

HAL\_CEC\_ERROR\_RXACKE

CEC Rx Missing Acknowledge

HAL\_CEC\_ERROR\_ARBLST

CEC Arbitration Lost

HAL\_CEC\_ERROR\_TXUDR

CEC Tx-Buffer Underrun

HAL\_CEC\_ERROR\_TXERR

CEC Tx-Error

HAL\_CEC\_ERROR\_TXACKE

CEC Tx Missing Acknowledge

***CEC Exported Macros***

**\_\_HAL\_CEC\_RESET\_HANDLE\_STATE**

**Description:**

- Reset CEC handle gstate & RxState.

**Parameters:**

- `__HANDLE__`: CEC handle.

**Return value:**

- None

## \_\_HAL\_CEC\_GET\_FLAG

### Description:

- Checks whether or not the specified CEC interrupt flag is set.

### Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__FLAG__`: specifies the flag to check.
  - `CEC_FLAG_TXACKE`: Tx Missing acknowledge Error
  - `CEC_FLAG_TXERR`: Tx Error.
  - `CEC_FLAG_TXUDR`: Tx-Buffer Underrun.
  - `CEC_FLAG_TXEND`: End of transmission (successful transmission of the last byte).
  - `CEC_FLAG_TXBR`: Tx-Byte Request.
  - `CEC_FLAG_ARBLST`: Arbitration Lost
  - `CEC_FLAG_RXACKE`: Rx-Missing Acknowledge
  - `CEC_FLAG_LBPE`: Rx Long period Error
  - `CEC_FLAG_SBPE`: Rx Short period Error
  - `CEC_FLAG_BRE`: Rx Bit Rising Error
  - `CEC_FLAG_RXOVR`: Rx Overrun.
  - `CEC_FLAG_RXEND`: End Of Reception.
  - `CEC_FLAG_RXBR`: Rx-Byte Received.

### Return value:

- `ITStatus`

## \_\_HAL\_CEC\_CLEAR\_FLAG

### Description:

- Clears the interrupt or status flag when raised (write at 1)

### Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__FLAG__`: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
  - `CEC_FLAG_TXACKE`: Tx Missing acknowledge Error
  - `CEC_FLAG_TXERR`: Tx Error.
  - `CEC_FLAG_TXUDR`: Tx-Buffer Underrun.
  - `CEC_FLAG_TXEND`: End of transmission (successful transmission of the last byte).
  - `CEC_FLAG_TXBR`: Tx-Byte Request.
  - `CEC_FLAG_ARBLST`: Arbitration Lost
  - `CEC_FLAG_RXACKE`: Rx-Missing Acknowledge
  - `CEC_FLAG_LBPE`: Rx Long period Error
  - `CEC_FLAG_SBPE`: Rx Short period Error
  - `CEC_FLAG_BRE`: Rx Bit Rising Error
  - `CEC_FLAG_RXOVR`: Rx Overrun.
  - `CEC_FLAG_RXEND`: End Of Reception.
  - `CEC_FLAG_RXBR`: Rx-Byte Received.

### Return value:

- `none`



## \_\_HAL\_CEC\_ENABLE\_IT

**Description:**

- Enables the specified CEC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to enable. This parameter can be one of the following values:
  - `CEC_IT_TXACK`: Tx Missing acknowledge Error IT Enable
  - `CEC_IT_TXERR`: Tx Error IT Enable
  - `CEC_IT_TXUDR`: Tx-Buffer Underrun IT Enable
  - `CEC_IT_TXEND`: End of transmission IT Enable
  - `CEC_IT_TXBR`: Tx-Byte Request IT Enable
  - `CEC_IT_ARBLST`: Arbitration Lost IT Enable
  - `CEC_IT_RXACK`: Rx-Missing Acknowledge IT Enable
  - `CEC_IT_LBPE`: Rx Long period Error IT Enable
  - `CEC_IT_SBPE`: Rx Short period Error IT Enable
  - `CEC_IT_BRE`: Rx Bit Rising Error IT Enable
  - `CEC_IT_RXOVR`: Rx Overrun IT Enable
  - `CEC_IT_RXEND`: End Of Reception IT Enable
  - `CEC_IT_RXBR`: Rx-Byte Received IT Enable

**Return value:**

- none

## \_\_HAL\_CEC\_DISABLE\_IT

**Description:**

- Disables the specified CEC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to disable. This parameter can be one of the following values:
  - `CEC_IT_TXACK`: Tx Missing acknowledge Error IT Enable
  - `CEC_IT_TXERR`: Tx Error IT Enable
  - `CEC_IT_TXUDR`: Tx-Buffer Underrun IT Enable
  - `CEC_IT_TXEND`: End of transmission IT Enable
  - `CEC_IT_TXBR`: Tx-Byte Request IT Enable
  - `CEC_IT_ARBLST`: Arbitration Lost IT Enable
  - `CEC_IT_RXACK`: Rx-Missing Acknowledge IT Enable
  - `CEC_IT_LBPE`: Rx Long period Error IT Enable
  - `CEC_IT_SBPE`: Rx Short period Error IT Enable
  - `CEC_IT_BRE`: Rx Bit Rising Error IT Enable
  - `CEC_IT_RXOVR`: Rx Overrun IT Enable
  - `CEC_IT_RXEND`: End Of Reception IT Enable
  - `CEC_IT_RXBR`: Rx-Byte Received IT Enable

**Return value:**

- none

### **\_\_HAL\_CEC\_GET\_IT\_SOURCE**

**Description:**

- Checks whether or not the specified CEC interrupt is enabled.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to check. This parameter can be one of the following values:
  - `CEC_IT_TXACK`: Tx Missing acknowledge Error IT Enable
  - `CEC_IT_TXERR`: Tx Error IT Enable
  - `CEC_IT_TXUDR`: Tx-Buffer Underrun IT Enable
  - `CEC_IT_TXEND`: End of transmission IT Enable
  - `CEC_IT_TXBR`: Tx-Byte Request IT Enable
  - `CEC_IT_ARBLST`: Arbitration Lost IT Enable
  - `CEC_IT_RXACK`: Rx-Missing Acknowledge IT Enable
  - `CEC_IT_LBPE`: Rx Long period Error IT Enable
  - `CEC_IT_SBPE`: Rx Short period Error IT Enable
  - `CEC_IT_BRE`: Rx Bit Rising Error IT Enable
  - `CEC_IT_RXOVR`: Rx Overrun IT Enable
  - `CEC_IT_RXEND`: End Of Reception IT Enable
  - `CEC_IT_RXBR`: Rx-Byte Received IT Enable

**Return value:**

- `FlagStatus`

### **\_\_HAL\_CEC\_ENABLE**

**Description:**

- Enables the CEC device.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- `none`

### **\_\_HAL\_CEC\_DISABLE**

**Description:**

- Disables the CEC device.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- `none`

### **\_\_HAL\_CEC\_FIRST\_BYTE\_TX\_SET**

**Description:**

- Set Transmission Start flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- `none`

### \_\_HAL\_CEC\_LAST\_BYTE\_TX\_SET

**Description:**

- Set Transmission End flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none: If the CEC message consists of only one byte, TXEOM must be set before of TXSOM.

### \_\_HAL\_CEC\_GET\_TRANSMISSION\_START\_FLAG

**Description:**

- Get Transmission Start flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- FlagStatus

### \_\_HAL\_CEC\_GET\_TRANSMISSION\_END\_FLAG

**Description:**

- Get Transmission End flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- FlagStatus

### \_\_HAL\_CEC\_CLEAR\_OAR

**Description:**

- Clear OAR register.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none

### \_\_HAL\_CEC\_SET\_OAR

**Description:**

- Set OAR register (without resetting previously set address in case of multi-address mode) To reset OAR,

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__ADDRESS__`: Own Address value (CEC logical address is identified by bit position)

**Return value:**

- none

**CEC Flags definition**

CEC\_FLAG\_TXACKE

CEC\_FLAG\_TXERR

CEC\_FLAG\_TXUDR

CEC\_FLAG\_TXEND

CEC\_FLAG\_TXBR

CEC\_FLAG\_ARBLST

CEC\_FLAG\_RXACKE

CEC\_FLAG\_LBPE

CEC\_FLAG\_SBPE

CEC\_FLAG\_BRE

CEC\_FLAG\_RXOVR

CEC\_FLAG\_RXEND

CEC\_FLAG\_RXBR

*CEC all RX errors interrupts enabling flag*

CEC\_IER\_RX\_ALL\_ERR

*CEC all TX errors interrupts enabling flag*

CEC\_IER\_TX\_ALL\_ERR

*CEC Initiator logical address position in message header*

CEC\_INITIATOR\_LSB\_POS

*CEC Interrupts definition*

CEC\_IT\_TXACKE

CEC\_IT\_TXERR

CEC\_IT\_TXUDR

CEC\_IT\_TXEND

CEC\_IT\_TXBR

CEC\_IT\_ARBLST

CEC\_IT\_RXACKE

CEC\_IT\_LBPE

CEC\_IT\_SBPE

CEC\_IT\_BRE

CEC\_IT\_RXOVR

CEC\_IT\_RXEND

CEC\_IT\_RXBR

*CEC Error Bit Generation if Long Bit Period Error reported*

CEC\_LBPE\_ERRORBIT\_NO\_GENERATION

CEC\_LBPE\_ERRORBIT\_GENERATION

*CEC Listening mode option*

CEC\_REDUCED\_LISTENING\_MODE

CEC\_FULL\_LISTENING\_MODE

*CEC Device Own Address position in CEC CFGR register*

CEC\_CFGR\_OAR\_LSB\_POS

*CEC Own Address*

CEC\_OWN\_ADDRESS\_NONE

CEC\_OWN\_ADDRESS\_0

CEC\_OWN\_ADDRESS\_1

CEC\_OWN\_ADDRESS\_2

CEC\_OWN\_ADDRESS\_3

CEC\_OWN\_ADDRESS\_4

CEC\_OWN\_ADDRESS\_5

CEC\_OWN\_ADDRESS\_6

CEC\_OWN\_ADDRESS\_7

CEC\_OWN\_ADDRESS\_8

CEC\_OWN\_ADDRESS\_9

CEC\_OWN\_ADDRESS\_10

CEC\_OWN\_ADDRESS\_11

CEC\_OWN\_ADDRESS\_12

CEC\_OWN\_ADDRESS\_13

CEC\_OWN\_ADDRESS\_14

*CEC Signal Free Time start option*

CEC\_SFT\_START\_ON\_TXSOM

CEC\_SFT\_START\_ON\_TX\_RX\_END

*CEC Signal Free Time setting parameter*

CEC\_DEFAULT\_SFT

CEC\_0\_5\_BITPERIOD\_SFT

CEC\_1\_5\_BITPERIOD\_SFT

CEC\_2\_5\_BITPERIOD\_SFT

CEC\_3\_5\_BITPERIOD\_SFT

CEC\_4\_5\_BITPERIOD\_SFT

CEC\_5\_5\_BITPERIOD\_SFT

CEC\_6\_5\_BITPERIOD\_SFT

**CEC State Code Definition**

HAL\_CEC\_STATE\_RESET

Peripheral is not yet Initialized Value is allowed for gState and RxState

HAL\_CEC\_STATE\_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL\_CEC\_STATE\_BUSY

an internal process is ongoing Value is allowed for gState only

HAL\_CEC\_STATE\_BUSY\_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL\_CEC\_STATE\_BUSY\_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL\_CEC\_STATE\_BUSY\_RX\_TX

an internal process is ongoing Value is allowed for gState only

HAL\_CEC\_STATE\_ERROR

Error Value is allowed for gState only

**CEC Receiver Tolerance**

CEC\_STANDARD\_TOLERANCE

CEC\_EXTENDED\_TOLERANCE

## 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 `stm32f4xx_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 according to the following table.
2. Configure the priority of the selected IRQ Channels using `HAL_NVIC_SetPriority()`.
3. Enable the selected IRQ Channels using `HAL_NVIC_EnableIRQ()`.
4. please refer to programming manual for details in how to configure priority.

*Note:* When the `NVIC_PRIORITYGROUP_0` is selected, IRQ preemption 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 preemption 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 `stm32f4xx_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 de-initialization 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\\_MPU\\_Disable\(\)](#)
- [HAL\\_MPU\\_Enable\(\)](#)
- [HAL\\_MPU\\_ConfigRegion\(\)](#)
- [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\(\)](#)

### 11.2.4 Detailed description of functions

#### HAL\_NVIC\_SetPriorityGrouping

##### Function name

```
void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)
```

##### Function description

Sets the priority grouping field (preemption 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 bits for preemption priority 4 bits for subpriority
  - NVIC\_PRIORITYGROUP\_1: 1 bits for preemption priority 3 bits for subpriority
  - NVIC\_PRIORITYGROUP\_2: 2 bits for preemption priority 2 bits for subpriority
  - NVIC\_PRIORITYGROUP\_3: 3 bits for preemption priority 1 bits for subpriority
  - NVIC\_PRIORITYGROUP\_4: 4 bits for preemption priority 0 bits for subpriority

##### Return values

- **None:**

##### Notes

- When the NVIC\_PriorityGroup\_0 is selected, IRQ preemption 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

Sets 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 (stm32f4xxxx.h))
- **PreemptPriority:** The preemption 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

Enables 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 (stm32f4xxxx.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

Disables 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 (stm32f4xxxx.h))

### Return values

- **None:**

#### HAL\_NVIC\_SystemReset

### Function name

**void HAL\_NVIC\_SystemReset (void )**

### Function description

Initiates 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

Initializes the System Timer and its interrupt, and starts the System Tick Timer.

### 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

Gets 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

Gets 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 (stm32f4xxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
  - NVIC\_PRIORITYGROUP\_0: 0 bits for preemption priority 4 bits for subpriority
  - NVIC\_PRIORITYGROUP\_1: 1 bits for preemption priority 3 bits for subpriority
  - NVIC\_PRIORITYGROUP\_2: 2 bits for preemption priority 2 bits for subpriority
  - NVIC\_PRIORITYGROUP\_3: 3 bits for preemption priority 1 bits for subpriority
  - NVIC\_PRIORITYGROUP\_4: 4 bits for preemption priority 0 bits 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

Gets Pending Interrupt (reads the pending register in the NVIC and returns 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 (stm32f4xxxx.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

Sets 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 (stm32f4xxxx.h))

### Return values

- **None:**

### HAL\_NVIC\_ClearPendingIRQ

### Function name

**void HAL\_NVIC\_ClearPendingIRQ (IRQn\_Type IRQn)**

### Function description

Clears 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 (stm32f4xxxx.h))

### Return values

- **None:**

### HAL\_NVIC\_GetActive

### Function name

**uint32\_t HAL\_NVIC\_GetActive (IRQn\_Type IRQn)**

### Function description

Gets active interrupt ( reads the active register in NVIC and returns 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 (stm32f4xxxx.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**

Configures 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**

This function handles 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 access to 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**

Disables the MPU.

**Return values**

- **None:**

**HAL\_MPU\_ConfigRegion**

**Function name**

**void HAL\_MPU\_ConfigRegion (MPU\_Region\_InitTypeDef \* MPU\_Init)**

**Function description**

Initializes and configures 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**

*MPU HFNMI and PRIVILEGED Access control*

**MPU\_HFNMI\_PRIVDEF\_NONE**

**MPU\_HARDFFAULT\_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

***MPU TEX Levels***

MPU\_TEX\_LEVEL0

MPU\_TEX\_LEVEL1

MPU\_TEX\_LEVEL2

***CORTEX Preemption Priority Group***

NVIC\_PRIORITYGROUP\_0

0 bits for pre-emption priority 4 bits for subpriority



**NVIC\_PRIORITYGROUP\_1**

1 bits 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 bits for subpriority

**NVIC\_PRIORITYGROUP\_4**

4 bits for pre-emption priority 0 bits 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\_HandleTypeDef

*CRC\_HandleTypeDef* is defined in the `stm32f4xx_hal_crc.h`

Data Fields

- *CRC\_TypeDef \* Instance*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_CRC\_StateTypeDef State*

Field Documentation

- *CRC\_TypeDef\* CRC\_HandleTypeDef::Instance*  
Register base address
- *HAL\_LockTypeDef CRC\_HandleTypeDef::Lock*  
CRC Locking object
- *\_\_IO HAL\_CRC\_StateTypeDef CRC\_HandleTypeDef::State*  
CRC communication state

### 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 32-bit CRC value of a 32-bit data buffer using combination of the previous CRC value and the new one.

or

- compute the 32-bit CRC value of a 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 32-bit CRC value of a 32-bit data buffer starting with the previously computed CRC as initialization value.

#### Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer.
- **BufferLength:** input data buffer length (number of uint32\_t words).

#### Return values

- **uint32\_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

**HAL\_CRC\_Calculate**

#### Function name

**uint32\_t HAL\_CRC\_Calculate (CRC\_HandleTypeDef \* hcrc, uint32\_t pBuffer, uint32\_t BufferLength)**

#### Function description

Compute the 32-bit CRC value of a 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

#### Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer.
- **BufferLength:** input data buffer length (number of uint32\_t words).

#### Return values

- **uint32\_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

**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 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\_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

## 13 HAL CRYP Generic Driver

### 13.1 CRYP Firmware driver registers structures

#### 13.1.1 CRYP\_ConfigTypeDef

*CRYP\_ConfigTypeDef* is defined in the `stm32f4xx_hal_cryp.h`

##### Data Fields

- *uint32\_t* *DataType*
- *uint32\_t* *KeySize*
- *uint32\_t* \* *pKey*
- *uint32\_t* \* *plnitVect*
- *uint32\_t* *Algorithm*
- *uint32\_t* \* *Header*
- *uint32\_t* *HeaderSize*
- *uint32\_t* \* *B0*
- *uint32\_t* *DataWidthUnit*
- *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::plnitVect*  
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 in word
- *uint32\_t*\* *CRYP\_ConfigTypeDef::B0*  
B0 is first authentication block used only in AES CCM mode
- *uint32\_t* *CRYP\_ConfigTypeDef::DataWidthUnit*  
Data With Unit, this parameter can be value of [CRYP\\_Data\\_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](#)

#### 13.1.2 \_\_CRYP\_HandleTypeDef

*\_\_CRYP\_HandleTypeDef* is defined in the `stm32f4xx_hal_cryp.h`

##### Data Fields

- *CRYP\_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\_STATTypeDef* State
- *\_\_IO uint32\_t* ErrorCode
- *uint32\_t* KeyIVConfig
- *uint32\_t* SizesSum

#### Field Documentation

- **CRYP\_TypeDef\* \_\_CRYP\_HandleTypeDef::Instance**  
CRYP registers base address
- **CRYP\_ConfigTypeDef \_\_CRYP\_HandleTypeDef::Init**  
CRYP required parameters
- **FunctionalState \_\_CRYP\_HandleTypeDef::AutoKeyDerivation**  
Used only in TinyAES to allows 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 word
- **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\_STATTypeDef \_\_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

## 13.2 CRYP Firmware driver API description

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

### 13.2.1 How to use this driver

The CRYP HAL driver can be used in CRYP or TinyAES IP 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 IP)
  - 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.
  - 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 parameters, 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 IP:
  - 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 IP:
  - 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 IP: 128-bit, 192-bit and 256-bit.
    - for TinyAES IP: 128-bit and 256-bit

This section describes the AES Galois/counter mode (GCM) supported by both CRYP1 IP:

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: IP prepares the GCM hash subkey (H) and do the IV processing
  - b. Header phase: IP processes the Additional Authenticated Data (AAD), with hash computation only.
  - c. Payload phase: IP processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
  - d. Final phase: IP 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)

This section describe The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) supported by both CRYP1 IP:

1. Specific parameters for CCM :
  - a. B0 block : According to NIST Special Publication 800-38C, The first block B0 is formatted as follows, where  $l(m)$  is encoded in most-significant-byte first order(see below table 3)
    - Q: a bit string representation of the octet length of P (plaintext)
    - q The octet length of the binary representation of the octet length of the payload
    - A nonce (N), n The octet length of the where  $n+q=15$ .
    - Flags: most significant octet containing four flags for control information,
    - t The octet length of the MAC.
  - b. B1 block (header) : associated data length(a) concatenated with Associated Data (A) the associated data length expressed in bytes (a) defined as below:
    - If  $0 < a < 216-28$ , then it is encoded as  $[a]16$ , i.e. two octets
    - If  $216-28 < a < 232$ , then it is encoded as  $0xff || 0xfe || [a]32$ , i.e. six octets
    - If  $232 < a < 264$ , then it is encoded as  $0xff || 0xff || [a]64$ , i.e. ten octets
  - c. CTRx block : control blocks
    - Generation of CTR1 from first block B0 information : equal to B0 with first 5 bits zeroed and most significant bits storing octet length of P also zeroed, then incremented by one ( see below Table 4)
    - Generation of CTR0: same as CTR1 with bit[0] set to zero.
2. Four phases are performed in CCM for CRYP1 IP:
  - a. Init phase: IP prepares the GCM hash subkey (H) and do the IV processing
  - b. Header phase: IP processes the Additional Authenticated Data (AAD), with hash computation only.
  - c. Payload phase: IP processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
  - d. Final phase: IP generates the authenticated tag (T) using the last block of data.

### Callback registration

## 13.2.2 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 IP : ECB and CBC in DES/TDES Standard ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard. - for TinyAES2 IP, 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\(\)\*](#)

## 13.2.3 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 IP, below list of Algorithm supported : - Electronic Code Book(ECB) - Cipher Block Chaining (CBC)
- Standard AES supported by CRYP1 IP & 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\(\)](#)

### 13.2.4 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\(\)](#)

### 13.2.5 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\_CRYPT\_GetConfig

### Function name

**HAL\_StatusTypeDef HAL\_CRYPT\_GetConfig (CRYP\_HandleTypeDef \* hcrypt, CRYPT\_ConfigTypeDef \* pConf)**

### Function description

Get CRYPT Configuration parameters in associated handle.

### Parameters

- **pConf:** pointer to a CRYPT\_ConfigTypeDef structure
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module

### Return values

- **HAL:** status

### HAL\_CRYPT\_Encrypt

### Function name

**HAL\_StatusTypeDef HAL\_CRYPT\_Encrypt (CRYP\_HandleTypeDef \* hcrypt, uint32\_t \* Input, uint16\_t Size, uint32\_t \* Output, uint32\_t Timeout)**

### Function description

Encryption mode.

### Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT 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\_CRYPT\_Decrypt

### Function name

**HAL\_StatusTypeDef HAL\_CRYPT\_Decrypt (CRYP\_HandleTypeDef \* hcrypt, uint32\_t \* Input, uint16\_t Size, uint32\_t \* Output, uint32\_t Timeout)**

### Function description

Decryption mode.

### Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT 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

- **hcrvp**: pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module.

### Return values

- **None**:

**HAL\_CRYP\_OutCpltCallback**

### Function name

**void HAL\_CRYP\_OutCpltCallback (CRYPT\_HandleTypeDef \* hcrvp)**

### Function description

Output FIFO transfer completed callback.

### Parameters

- **hcrvp**: pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module.

### Return values

- **None**:

**HAL\_CRYP\_ErrorCallback**

### Function name

**void HAL\_CRYP\_ErrorCallback (CRYPT\_HandleTypeDef \* hcrvp)**

### Function description

CRYP error callback.

### Parameters

- **hcrvp**: pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module.

### Return values

- **None**:

**HAL\_CRYP\_GetError**

### Function name

**uint32\_t HAL\_CRYP\_GetError (CRYPT\_HandleTypeDef \* hcrvp)**

### Function description

Return the CRYPT error code.

### Parameters

- **hcrvp**: : pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for the CRYPT IP

### Return values

- **CRYP**: error code

## 13.3 CRYP Firmware driver defines

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



**13.3.1 CRYPT**  
CRYPT  
**CRYP Algorithm Mode**

CRYP\_DES\_ECB

CRYP\_DES\_CBC

CRYP\_TDES\_ECB

CRYP\_TDES\_CBC

CRYP\_AES\_ECB

CRYP\_AES\_CBC

CRYP\_AES\_CTR

CRYP\_AES\_GCM

CRYP\_AES\_CCM

**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

CRYP\_DATATYPE\_16B

CRYP\_DATATYPE\_8B

CRYP\_DATATYPE\_1B

**CRYP Data Width Unit**

CRYP\_DATAWIDTHUNIT\_WORD

By default, size unit is word

CRYP\_DATAWIDTHUNIT\_BYTE

By default, size unit is word

**CRYP Error Definition**

HAL\_CRYPT\_ERROR\_NONE

No error

HAL\_CRYPT\_ERROR\_WRITE

Write error

HAL\_CRYPT\_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:

- `__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\_GET\_IT

### Description:

- Clear the CRYP pending status flag.

### Parameters:

- `__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
- `__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
- `__HANDLE__`: specifies the CRYP handle.

### Return value:

- NoneCheck: whether the specified CRYP interrupt is set or not.
- The: state of `__INTERRUPT__` (TRUE or FALSE).

### \_\_HAL\_CRYP\_ENABLE\_IT

**Description:**

- Enable the CRYP interrupt.

**Parameters:**

- `__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.
- `__HANDLE__`: specifies the CRYP handle.

**Return value:**

- None

### \_\_HAL\_CRYP\_DISABLE\_IT

**Description:**

- Disable the CRYP interrupt.

**Parameters:**

- `__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.
- `__HANDLE__`: specifies the CRYP handle.

**Return value:**

- None

**CRYP Flags**

#### CRYP\_FLAG\_IFEM

Input FIFO is empty

#### CRYP\_FLAG\_IFNF

Input FIFO is not Full

#### CRYP\_FLAG\_OFNE

Output FIFO is not empty

#### CRYP\_FLAG\_OFFU

Output FIFO is Full

#### CRYP\_FLAG\_BUSY

The CRYP core is currently processing a block of data or a key preparation (for AES decryption).

#### CRYP\_FLAG\_OUTRIS

Output FIFO service raw interrupt status

#### CRYP\_FLAG\_INRIS

Input FIFO service raw interrupt status

**CRYP Interrupt**

#### CRYP\_IT\_INI

Input FIFO Interrupt

#### CRYP\_IT\_OUTI

Output FIFO Interrupt

*CRYP Private macros to check input parameters*

IS\_CRYP\_ALGORITHM

IS\_CRYP\_KEYSIZE

IS\_CRYP\_DATATYPE

IS\_CRYP\_INIT

*CRYP Key Size*

CRYP\_KEYSIZE\_128B

CRYP\_KEYSIZE\_192B

CRYP\_KEYSIZE\_256B

## 14 HAL CRYPT Extension Driver

### 14.1 CRYPEX Firmware driver API description

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

#### 14.1.1 How to use this driver

The CRYPT extension HAL driver can be used as follows:

1. After AES-GCM or AES-CCM Encryption/Decryption user can start following API to get the authentication messages :
  - a. HAL\_CRYPEX\_AESGCM\_GenerateAuthTAG
  - b. HAL\_CRYPEX\_AESCCM\_GenerateAuthTAG

#### 14.1.2 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\(\)](#)

#### 14.1.3 Detailed description of functions

##### HAL\_CRYPEX\_AESGCM\_GenerateAuthTAG

###### Function name

**HAL\_StatusTypeDef HAL\_CRYPEX\_AESGCM\_GenerateAuthTAG (CRYPT\_HandleTypeDef \* hcrypt, uint32\_t \* AuthTag, uint32\_t Timeout)**

###### Function description

generate the GCM authentication TAG.

###### Parameters

- **hcryp**: pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT 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 (CRYPT\_HandleTypeDef \* hcrypt, uint32\_t \* AuthTag, uint32\_t Timeout)**

###### Function description

AES CCM Authentication TAG generation.

### Parameters

- **hcryp**: pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
- **AuthTag**: Pointer to the authentication buffer
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

## 15 HAL DAC Generic Driver

### 15.1 DAC Firmware driver registers structures

#### 15.1.1 DAC\_HandleTypeDef

*DAC\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 15.1.2 DAC\_ChannelConfTypeDef

*DAC\_ChannelConfTypeDef* is defined in the `stm32f4xx_hal_dac.h`

##### Data Fields

- *uint32\_t DAC\_Trigger*
- *uint32\_t DAC\_OutputBuffer*

##### Field Documentation

- *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](#)
- *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](#)

### 15.2 DAC Firmware driver API description

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

#### 15.2.1 DAC Peripheral features

##### DAC Channels

The device integrates two 12-bit Digital Analog Converters that can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC\_OUT1 (PA4) as output



2. DAC channel2 with DAC\_OUT2 (PA5) as output

### 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\_Pin9) using DAC\_TRIGGER\_EXT\_IT9. The used pin (GPIOx\_Pin9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM4, TIM5, TIM6, TIM7 and TIM8 (DAC\_TRIGGER\_T2\_TRGO, DAC\_TRIGGER\_T4\_TRGO...)
3. Software using DAC\_TRIGGER\_SOFTWARE

### 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 wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle 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 DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using `HAL_DAC_Start_DMA()`

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 Stream5 channel7 which must be already configured
2. DAC channel2 : mapped on DMA1 Stream6 channel7 which must be already configured

*Note:* For Dual mode and specific signal (Triangle and noise) generation please refer to Extension Features Driver description

## 15.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using `HAL_DAC_Init()`
- Configure DAC\_OUTx (DAC\_OUT1: PA4, DAC\_OUT2: PA5) 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

### 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
- At The end of data transfer HAL\_DAC\_ConvCpltCallbackCh1() or HAL\_DAC\_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ConvCpltCallbackCh1 or HAL\_DAC\_ConvCpltCallbackCh2
- 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
- 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/Delnit callbacks can be used during the Init/Delnit. 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

### 15.2.3 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\(\)\*](#)

### 15.2.4 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\\_GetValue\(\)\*](#)
- [\*HAL\\_DAC\\_IRQHandler\(\)\*](#)
- [\*HAL\\_DAC\\_ConvCpltCallbackCh1\(\)\*](#)
- [\*HAL\\_DAC\\_ConvHalfCpltCallbackCh1\(\)\*](#)
- [\*HAL\\_DAC\\_ErrorCallbackCh1\(\)\*](#)
- [\*HAL\\_DAC\\_DMAUnderrunCallbackCh1\(\)\*](#)

### 15.2.5 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\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_DAC\\_SetValue\(\)\*](#)

### 15.2.6 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\(\)](#)
- [HAL\\_DAC\\_IRQHandler\(\)](#)
- [HAL\\_DAC\\_ConvCpltCallbackCh1\(\)](#)
- [HAL\\_DAC\\_ConvHalfCpltCallbackCh1\(\)](#)
- [HAL\\_DAC\\_ErrorCallbackCh1\(\)](#)
- [HAL\\_DAC\\_DMAUnderrunCallbackCh1\(\)](#)

### 15.2.7 Detailed description of functions

#### HAL\_DAC\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_DAC\_Init (DAC\_HandleTypeDef \* hdac)**

##### Function description

Initializes the DAC peripheral according to the specified parameters in the DAC\_InitStruct.

##### 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

Deinitializes 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

Initializes 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

Deinitializes 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

#### 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

#### 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
- **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

### Return values

- **HAL:** status

### 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

### 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

### Return values

- **HAL**: status

## 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
- **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\_GetState

### Function name

HAL\_DAC\_StateTypeDef HAL\_DAC\_GetState (DAC\_HandleTypeDef \* hdac)

### Function description

return the DAC state

### Parameters

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.

**Return values**

- **HAL:** state

**HAL\_DAC\_IRQHandler**
**Function name**
**void HAL\_DAC\_IRQHandler (DAC\_HandleTypeDef \* hdac)**
**Function description**

Handles DAC interrupt request.

**Parameters**

- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.

**Return values**

- **None:**

**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

**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:**

## 15.3 DAC Firmware driver defines

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

### 15.3.1 DAC

DAC

***DAC Channel Selection***

**DAC\_CHANNEL\_1**

**DAC\_CHANNEL\_2**

***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 DAM underrun error

**HAL\_DAC\_ERROR\_DMAUNDERRUNCH2**

DAC channel2 DAM underrun error

**HAL\_DAC\_ERROR\_DMA**

DMA 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

**\_\_HAL\_DAC\_ENABLE\_IT**
**Description:**

- Enable the DAC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt.

**Return value:**

- None

### **\_\_HAL\_DAC\_DISABLE\_IT**

**Description:**

- Disable the DAC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt.

**Return value:**

- None

### **\_\_HAL\_DAC\_GET\_IT\_SOURCE**

**Description:**

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

**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

**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 flag to clear. This parameter can be any combination of the following values:
  - `DAC_FLAG_DMAUDR1`: DMA underrun 1 flag
  - `DAC_FLAG_DMAUDR2`: DMA underrun 2 flag

**Return value:**

- None

### **\_\_HAL\_DAC\_CLEAR\_FLAG**

**Description:**

- Clear the DAC's flag.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DAC_FLAG_DMAUDR1`: DMA underrun 1 flag
  - `DAC_FLAG_DMAUDR2`: DMA underrun 2 flag

**Return value:**

- None

***DAC Flags Definition***

**DAC\_FLAG\_DMAUDR1**

**DAC\_FLAG\_DMAUDR2**

***DAC IT Definition***

**DAC\_IT\_DMAUDR1**

**DAC\_IT\_DMAUDR2**

*DAC Output Buffer*

**DAC\_OUTPUTBUFFER\_ENABLE****DAC\_OUTPUTBUFFER\_DISABLE**

*DAC Trigger Selection*

**DAC\_TRIGGER\_NONE**

Conversion is automatic once the DAC1\_DHRxxxx register has been loaded, and not by external trigger

**DAC\_TRIGGER\_T2\_TRGO**

TIM2 TRGO selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_T4\_TRGO**

TIM4 TRGO selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_T5\_TRGO**

TIM5 TRGO selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_T6\_TRGO**

TIM6 TRGO selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_T7\_TRGO**

TIM7 TRGO selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_T8\_TRGO**

TIM8 TRGO selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_EXT\_IT9**

EXTI Line9 event selected as external conversion trigger for DAC channel

**DAC\_TRIGGER\_SOFTWARE**

Conversion started by software trigger for DAC channel

## 16 HAL DAC Extension Driver

### 16.1 DACEx Firmware driver API description

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

#### 16.1.1 How to use this driver

- When Dual mode is enabled (i.e DAC Channel1 and Channel2 are used simultaneously) : Use HAL\_DACEx\_DualGetValue() to get digital data to be converted and use HAL\_DACEx\_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.
- Use HAL\_DACEx\_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL\_DACEx\_NoiseWaveGenerate() to generate Noise signal.

#### 16.1.2 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\\_DualGetValue\(\)](#)
- [HAL\\_DACEx\\_TriangleWaveGenerate\(\)](#)
- [HAL\\_DACEx\\_NoiseWaveGenerate\(\)](#)
- [HAL\\_DACEx\\_DualSetValue\(\)](#)
- [HAL\\_DACEx\\_ConvCpltCallbackCh2\(\)](#)
- [HAL\\_DACEx\\_ConvHalfCpltCallbackCh2\(\)](#)
- [HAL\\_DACEx\\_ErrorCallbackCh2\(\)](#)
- [HAL\\_DACEx\\_DMAUnderrunCallbackCh2\(\)](#)

#### 16.1.3 Detailed description of functions

##### HAL\_DACEx\_DualGetValue

###### Function name

`uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)`

###### 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.

###### Return values

- **The**: selected DAC channel data output value.

##### HAL\_DACEx\_TriangleWaveGenerate

###### Function name

`HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)`

## Function description

Enables or disables 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\_CHANNEL\_2
- **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

Enables or disables 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\_CHANNEL\_2
- **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\_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 Channel2 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel1 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\_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:**

### 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:**

## 16.2 DACEx Firmware driver defines

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

### 16.2.1 DACEx

DACEx

***DAC LFS Run Mask 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

## 17 HAL DCMI Generic Driver

### 17.1 DCMI Firmware driver registers structures

#### 17.1.1 DCMI\_SyncUnmaskTypeDef

*DCMI\_SyncUnmaskTypeDef* is defined in the `stm32f4xx_hal_dcmi.h`

##### Data Fields

- *uint8\_t FrameStartUnmask*
- *uint8\_t LineStartUnmask*
- *uint8\_t LineEndUnmask*
- *uint8\_t FrameEndUnmask*

##### Field Documentation

- *uint8\_t DCMI\_SyncUnmaskTypeDef::FrameStartUnmask*  
Specifies the frame start delimiter unmask.
- *uint8\_t DCMI\_SyncUnmaskTypeDef::LineStartUnmask*  
Specifies the line start delimiter unmask.
- *uint8\_t DCMI\_SyncUnmaskTypeDef::LineEndUnmask*  
Specifies the line end delimiter unmask.
- *uint8\_t DCMI\_SyncUnmaskTypeDef::FrameEndUnmask*  
Specifies the frame end delimiter unmask.

#### 17.1.2 \_\_DCMI\_HandleTypeDef

*\_\_DCMI\_HandleTypeDef* is defined in the `stm32f4xx_hal_dcmi.h`

##### Data Fields

- *DCMI\_TypeDef \* Instance*
- *DCMI\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_DCMI\_StateTypeDef State*
- *\_\_IO uint32\_t XferCount*
- *\_\_IO uint32\_t XferSize*
- *uint32\_t XferTransferNumber*
- *uint32\_t pBuffPtr*
- *DMA\_HandleTypeDef \* DMA\_Handle*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- *DCMI\_TypeDef\* \_\_DCMI\_HandleTypeDef::Instance*  
DCMI Register base address
- *DCMI\_InitTypeDef \_\_DCMI\_HandleTypeDef::Init*  
DCMI parameters
- *HAL\_LockTypeDef \_\_DCMI\_HandleTypeDef::Lock*  
DCMI locking object
- *\_\_IO HAL\_DCMI\_StateTypeDef \_\_DCMI\_HandleTypeDef::State*  
DCMI state
- *\_\_IO uint32\_t \_\_DCMI\_HandleTypeDef::XferCount*  
DMA transfer counter
- *\_\_IO uint32\_t \_\_DCMI\_HandleTypeDef::XferSize*  
DMA transfer size

- **`uint32_t __DCMI_HandleTypeDef::XferTransferNumber`**  
DMA transfer number
- **`uint32_t __DCMI_HandleTypeDef::pBuffPtr`**  
Pointer to DMA output buffer
- **`DMA_HandleTypeDef* __DCMI_HandleTypeDef::DMA_Handle`**  
Pointer to the DMA handler
- **`__IO uint32_t __DCMI_HandleTypeDef::ErrorCode`**  
DCMI Error code

## 17.2 DCMI Firmware driver API description

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

### 17.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using `HAL_DCMI_Init()` function.
2. Configure the DMA2\_Stream1 channel1 to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using `HAL_DCMI_Start_DMA()` function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using `HAL_DCMI_ConfigCrop()` and `HAL_DCMI_EnableCROP()` functions
5. The capture can be stopped using `HAL_DCMI_Stop()` function.
6. To control DCMI state you can use the function `HAL_DCMI_GetState()`.

#### DCMI HAL driver macros list

Below the list of most used macros in DCMI HAL driver.

- `__HAL_DCMI_ENABLE`: Enable the DCMI peripheral.
- `__HAL_DCMI_DISABLE`: Disable the DCMI peripheral.
- `__HAL_DCMI_GET_FLAG`: Get the DCMI pending flags.
- `__HAL_DCMI_CLEAR_FLAG`: Clear the DCMI pending flags.
- `__HAL_DCMI_ENABLE_IT`: Enable the specified DCMI interrupts.
- `__HAL_DCMI_DISABLE_IT`: Disable the specified DCMI interrupts.
- `__HAL_DCMI_GET_IT_SOURCE`: Check whether the specified DCMI interrupt has occurred or not.

*Note:* You can refer to the DCMI HAL driver header file for more useful macros

#### Callback registration

### 17.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- **`HAL_DCMI_Init()`**
- **`HAL_DCMI_DeInit()`**
- **`HAL_DCMI_MspInit()`**
- **`HAL_DCMI_MspDeInit()`**

### 17.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- Handles DCMI interrupt request.

This section contains the following APIs:

- [\*HAL\\_DCMI\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_DCMI\\_Stop\(\)\*](#)
- [\*HAL\\_DCMI\\_Suspend\(\)\*](#)
- [\*HAL\\_DCMI\\_Resume\(\)\*](#)
- [\*HAL\\_DCMI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_DCMI\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_LineEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_VsyncEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_FrameEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_VsyncCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_HsyncCallback\(\)\*](#)

### 17.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.

This section contains the following APIs:

- [\*HAL\\_DCMI\\_ConfigCrop\(\)\*](#)
- [\*HAL\\_DCMI\\_DisableCrop\(\)\*](#)
- [\*HAL\\_DCMI\\_EnableCrop\(\)\*](#)
- [\*HAL\\_DCMI\\_ConfigSyncUnmask\(\)\*](#)

### 17.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:

- [\*HAL\\_DCMI\\_GetState\(\)\*](#)
- [\*HAL\\_DCMI\\_GetError\(\)\*](#)

### 17.2.6 Detailed description of functions

#### HAL\_DCMI\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_DCMI\_Init (DCMI\_HandleTypeDef \* hdcmi)**

##### Function description

Initializes the DCMI according to the specified parameters in the DCMI\_InitTypeDef and create the associated handle.

##### Parameters

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **HAL:** status

**HAL\_DCMI\_DeInit**

**Function name**

**HAL\_StatusTypeDef HAL\_DCMI\_DeInit (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Deinitializes the DCMI peripheral registers to their default reset values.

**Parameters**

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **HAL:** status

**HAL\_DCMI\_MspInit**

**Function name**

**void HAL\_DCMI\_MspInit (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Initializes the DCMI MSP.

**Parameters**

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **None:**

**HAL\_DCMI\_MspDeInit**

**Function name**

**void HAL\_DCMI\_MspDeInit (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Deinitializes the DCMI MSP.

**Parameters**

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **None:**

**HAL\_DCMI\_Start\_DMA**

**Function name**

**HAL\_StatusTypeDef HAL\_DCMI\_Start\_DMA (DCMI\_HandleTypeDef \* hdcmi, uint32\_t DCMI\_Mode, uint32\_t pData, uint32\_t Length)**

**Function description**

Enables DCMI DMA request and enables DCMI capture.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- **DCMI\_Mode**: DCMI capture mode snapshot or continuous grab.
- **pData**: The destination memory Buffer address (LCD Frame buffer).
- **Length**: The length of capture to be transferred.

**Return values**

- **HAL**: status

**HAL\_DCMI\_Stop**
**Function name**

**HAL\_StatusTypeDef HAL\_DCMI\_Stop (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Disable DCMI DMA request and Disable DCMI capture.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **HAL**: status

**HAL\_DCMI\_Suspend**
**Function name**

**HAL\_StatusTypeDef HAL\_DCMI\_Suspend (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Suspend DCMI capture.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **HAL**: status

**HAL\_DCMI\_Resume**
**Function name**

**HAL\_StatusTypeDef HAL\_DCMI\_Resume (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Resume DCMI capture.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **HAL**: status

**HAL\_DCMI\_ErrorCallback**
**Function name**

**void HAL\_DCMI\_ErrorCallback (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Error DCMI callback.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **None**:

**HAL\_DCMI\_LineEventCallback**

**Function name**

**void HAL\_DCMI\_LineEventCallback (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Line Event callback.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **None**:

**HAL\_DCMI\_FrameEventCallback**

**Function name**

**void HAL\_DCMI\_FrameEventCallback (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

Frame Event callback.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **None**:

**HAL\_DCMI\_VsyncEventCallback**

**Function name**

**void HAL\_DCMI\_VsyncEventCallback (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**

VSYNC Event callback.

**Parameters**

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

**Return values**

- **None**:

**HAL\_DCMI\_VsyncCallback**

**Function name**

**void HAL\_DCMI\_VsyncCallback (DCMI\_HandleTypeDef \* hdcmi)**

**Function description**



### HAL\_DCMI\_HsyncCallback

#### Function name

**void HAL\_DCMI\_HsyncCallback (DCMI\_HandleTypeDef \* hdcmi)**

#### Function description

### HAL\_DCMI\_IRQHandler

#### Function name

**void HAL\_DCMI\_IRQHandler (DCMI\_HandleTypeDef \* hdcmi)**

#### Function description

Handles DCMI interrupt request.

#### Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for the DCMI.

#### Return values

- **None:**

### HAL\_DCMI\_ConfigCrop

#### Function name

**HAL\_StatusTypeDef HAL\_DCMI\_ConfigCrop (DCMI\_HandleTypeDef \* hdcmi, uint32\_t X0, uint32\_t Y0, uint32\_t XSize, uint32\_t YSize)**

#### Function description

Configure the DCMI CROP coordinate.

#### Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- **X0:** DCMI window X offset
- **Y0:** DCMI window Y offset
- **XSize:** DCMI Pixel per line
- **YSize:** DCMI Line number

#### Return values

- **HAL:** status

### HAL\_DCMI\_EnableCrop

#### Function name

**HAL\_StatusTypeDef HAL\_DCMI\_EnableCrop (DCMI\_HandleTypeDef \* hdcmi)**

#### Function description

Enable the Crop feature.

#### Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

#### Return values

- **HAL:** status

### HAL\_DCMI\_DisableCrop

#### Function name

HAL\_StatusTypeDef HAL\_DCMI\_DisableCrop (DCMI\_HandleTypeDef \* hdcmi)

#### Function description

Disable the Crop feature.

#### Parameters

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

#### Return values

- **HAL**: status

### HAL\_DCMI\_ConfigSyncUnmask

#### Function name

HAL\_StatusTypeDef HAL\_DCMI\_ConfigSyncUnmask (DCMI\_HandleTypeDef \* hdcmi, DCMI\_SyncUnmaskTypeDef \* SyncUnmask)

#### Function description

Set embedded synchronization delimiters unmask.

#### Parameters

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- **SyncUnmask**: pointer to a DCMI\_SyncUnmaskTypeDef structure that contains the embedded synchronization delimiters unmask.

#### Return values

- **HAL**: status

### HAL\_DCMI\_GetState

#### Function name

HAL\_DCMI\_StateTypeDef HAL\_DCMI\_GetState (DCMI\_HandleTypeDef \* hdcmi)

#### Function description

Return the DCMI state.

#### Parameters

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

#### Return values

- **HAL**: state

### HAL\_DCMI\_GetError

#### Function name

uint32\_t HAL\_DCMI\_GetError (DCMI\_HandleTypeDef \* hdcmi)

#### Function description

Return the DCMI error code.

#### Parameters

- **hdcmi**: pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

#### Return values

- **DCMI**: Error Code

## 17.3 DCMI Firmware driver defines

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

### 17.3.1 DCMI

DCMI

#### *DCMI Capture Mode*

#### DCMI\_MODE\_CONTINUOUS

The received data are transferred continuously into the destination memory through the DMA

#### DCMI\_MODE\_SNAPSHOT

Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

#### *DCMI Capture Rate*

#### DCMI\_CR\_ALL\_FRAME

All frames are captured

#### DCMI\_CR\_ALTERNATE\_2\_FRAME

Every alternate frame captured

#### DCMI\_CR\_ALTERNATE\_4\_FRAME

One frame in 4 frames captured

#### *DCMI Error Code*

#### HAL\_DCMI\_ERROR\_NONE

No error

#### HAL\_DCMI\_ERROR\_OVR

Overrun error

#### HAL\_DCMI\_ERROR\_SYNC

Synchronization error

#### HAL\_DCMI\_ERROR\_TIMEOUT

Timeout error

#### HAL\_DCMI\_ERROR\_DMA

DMA error

#### *DCMI Exported Macros*

#### \_\_HAL\_DCMI\_RESET\_HANDLE\_STATE

##### **Description:**

- Reset DCMI handle state.

##### **Parameters:**

- `__HANDLE__`: specifies the DCMI handle.

##### **Return value:**

- None

**\_\_HAL\_DCMI\_ENABLE**
**Description:**

- Enable the DCMI.

**Parameters:**

- `__HANDLE__`: DCMI handle

**Return value:**

- None

**\_\_HAL\_DCMI\_DISABLE**
**Description:**

- Disable the DCMI.

**Parameters:**

- `__HANDLE__`: DCMI handle

**Return value:**

- None

**\_\_HAL\_DCMI\_GET\_FLAG**
**Description:**

- Get the DCMI pending flag.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__FLAG__`: Get the specified flag. This parameter can be one of the following values (no combination allowed)
  - `DCMI_FLAG_HSYNC`: HSYNC pin state (active line / synchronization between lines)
  - `DCMI_FLAG_VSYNC`: VSYNC pin state (active frame / synchronization between frames)
  - `DCMI_FLAG_FNE`: FIFO empty flag
  - `DCMI_FLAG_FRAMERI`: Frame capture complete flag mask
  - `DCMI_FLAG_OVRRI`: Overrun flag mask
  - `DCMI_FLAG_ERRRI`: Synchronization error flag mask
  - `DCMI_FLAG_VSYNCR`: VSYNC flag mask
  - `DCMI_FLAG_LINER`: Line flag mask
  - `DCMI_FLAG_FRAMEMI`: DCMI Capture complete masked interrupt status
  - `DCMI_FLAG_OVRMI`: DCMI Overrun masked interrupt status
  - `DCMI_FLAG_ERRMI`: DCMI Synchronization error masked interrupt status
  - `DCMI_FLAG_VSYNCMI`: DCMI VSYNC masked interrupt status
  - `DCMI_FLAG_LINEMI`: DCMI Line masked interrupt status

**Return value:**

- The: state of FLAG.

### **\_\_HAL\_DCMI\_CLEAR\_FLAG**

**Description:**

- Clear the DCMI pending flags.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DCMI_FLAG_FRAMERI`: Frame capture complete flag mask
  - `DCMI_FLAG_OVRRRI`: Overrun flag mask
  - `DCMI_FLAG_ERRRI`: Synchronization error flag mask
  - `DCMI_FLAG_VSYNCR`: VSYNC flag mask
  - `DCMI_FLAG_LINER`: Line flag mask

**Return value:**

- None

### **\_\_HAL\_DCMI\_ENABLE\_IT**

**Description:**

- Enable the specified DCMI interrupts.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__INTERRUPT__`: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - `DCMI_IT_FRAME`: Frame capture complete interrupt mask
  - `DCMI_IT_OVR`: Overrun interrupt mask
  - `DCMI_IT_ERR`: Synchronization error interrupt mask
  - `DCMI_IT_VSYNC`: VSYNC interrupt mask
  - `DCMI_IT_LINE`: Line interrupt mask

**Return value:**

- None

### **\_\_HAL\_DCMI\_DISABLE\_IT**

**Description:**

- Disable the specified DCMI interrupts.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__INTERRUPT__`: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - `DCMI_IT_FRAME`: Frame capture complete interrupt mask
  - `DCMI_IT_OVR`: Overrun interrupt mask
  - `DCMI_IT_ERR`: Synchronization error interrupt mask
  - `DCMI_IT_VSYNC`: VSYNC interrupt mask
  - `DCMI_IT_LINE`: Line interrupt mask

**Return value:**

- None

## **\_\_HAL\_DCMI\_GET\_IT\_SOURCE**

### **Description:**

- Check whether the specified DCMI interrupt has occurred or not.

### **Parameters:**

- `__HANDLE__`: DCMI handle
- `__INTERRUPT__`: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
  - `DCMI_IT_FRAME`: Frame capture complete interrupt mask
  - `DCMI_IT_OVR`: Overrun interrupt mask
  - `DCMI_IT_ERR`: Synchronization error interrupt mask
  - `DCMI_IT_VSYNC`: VSYNC interrupt mask
  - `DCMI_IT_LINE`: Line interrupt mask

### **Return value:**

- The: state of `INTERRUPT`.

### **DCMI Extended Data Mode**

#### **DCMI\_EXTEND\_DATA\_8B**

Interface captures 8-bit data on every pixel clock

#### **DCMI\_EXTEND\_DATA\_10B**

Interface captures 10-bit data on every pixel clock

#### **DCMI\_EXTEND\_DATA\_12B**

Interface captures 12-bit data on every pixel clock

#### **DCMI\_EXTEND\_DATA\_14B**

Interface captures 14-bit data on every pixel clock

### **DCMI Flags**

#### **DCMI\_FLAG\_HSYNC**

HSYNC pin state (active line / synchronization between lines)

#### **DCMI\_FLAG\_VSYNC**

VSYNC pin state (active frame / synchronization between frames)

#### **DCMI\_FLAG\_FNE**

FIFO not empty flag

#### **DCMI\_FLAG\_FRAMERI**

Frame capture complete interrupt flag

#### **DCMI\_FLAG\_OVRR**

Overrun interrupt flag

#### **DCMI\_FLAG\_ERRR**

Synchronization error interrupt flag

#### **DCMI\_FLAG\_VSYNCR**

VSYNC interrupt flag

#### **DCMI\_FLAG\_LINER**

Line interrupt flag

#### **DCMI\_FLAG\_FRAMEM**

DCMI Frame capture complete masked interrupt status

**DCMI\_FLAG\_OVRMI**

DCMI Overrun masked interrupt status

**DCMI\_FLAG\_ERRMI**

DCMI Synchronization error masked interrupt status

**DCMI\_FLAG\_VSYNCMI**

DCMI VSYNC masked interrupt status

**DCMI\_FLAG\_LINEMI**

DCMI Line masked interrupt status

***DCMI HSYNC Polarity***

**DCMI\_HSPOLARITY\_LOW**

Horizontal synchronization active Low

**DCMI\_HSPOLARITY\_HIGH**

Horizontal synchronization active High

***DCMI interrupt sources***

**DCMI\_IT\_FRAME**

Capture complete interrupt

**DCMI\_IT\_OVR**

Overrun interrupt

**DCMI\_IT\_ERR**

Synchronization error interrupt

**DCMI\_IT\_VSYNC**

VSYNC interrupt

**DCMI\_IT\_LINE**

Line interrupt

***DCMI MODE JPEG***

**DCMI\_JPEG\_DISABLE**

Mode JPEG Disabled

**DCMI\_JPEG\_ENABLE**

Mode JPEG Enabled

***DCMI PIXCK Polarity***

**DCMI\_PCKPOLARITY\_FALLING**

Pixel clock active on Falling edge

**DCMI\_PCKPOLARITY\_RISING**

Pixel clock active on Rising edge

***DCMI Synchronization Mode***

**DCMI\_SYNCHRO\_HARDWARE**

Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

**DCMI\_SYNCHRO\_EMBEDDED**

Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

***DCMI VSYNC Polarity***

**DCMI\_VSPOLARITY\_LOW**

Vertical synchronization active Low

**DCMI\_VSPOLARITY\_HIGH**

Vertical synchronization active High

***DCMI Window Coordinate***

**DCMI\_WINDOW\_COORDINATE**

Window coordinate

***DCMI Window Height***

**DCMI\_WINDOW\_HEIGHT**

Window Height

***DCMI Window Vertical Line***

**DCMI\_POSITION\_CWSIZE\_VLINE**

Required left shift to set crop window vertical line count

**DCMI\_POSITION\_CWSTRT\_VST**

Required left shift to set crop window vertical start line count



## 18 HAL DCMI Extension Driver

### 18.1 DCMIEx Firmware driver registers structures

#### 18.1.1 DCMI\_CodesInitTypeDef

*DCMI\_CodesInitTypeDef* is defined in the `stm32f4xx_hal_dcmi_ex.h`

##### Data Fields

- *uint8\_t FrameStartCode*
- *uint8\_t LineStartCode*
- *uint8\_t LineEndCode*
- *uint8\_t FrameEndCode*

##### Field Documentation

- *uint8\_t DCMI\_CodesInitTypeDef::FrameStartCode*  
Specifies the code of the frame start delimiter.
- *uint8\_t DCMI\_CodesInitTypeDef::LineStartCode*  
Specifies the code of the line start delimiter.
- *uint8\_t DCMI\_CodesInitTypeDef::LineEndCode*  
Specifies the code of the line end delimiter.
- *uint8\_t DCMI\_CodesInitTypeDef::FrameEndCode*  
Specifies the code of the frame end delimiter.

#### 18.1.2 DCMI\_InitTypeDef

*DCMI\_InitTypeDef* is defined in the `stm32f4xx_hal_dcmi_ex.h`

##### Data Fields

- *uint32\_t SynchroMode*
- *uint32\_t PCKPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t HSPolarity*
- *uint32\_t CaptureRate*
- *uint32\_t ExtendedDataMode*
- *DCMI\_CodesInitTypeDef SyncroCode*
- *uint32\_t JPEGMode*
- *uint32\_t ByteSelectMode*
- *uint32\_t ByteSelectStart*
- *uint32\_t LineSelectMode*
- *uint32\_t LineSelectStart*

##### Field Documentation

- *uint32\_t DCMI\_InitTypeDef::SynchroMode*  
Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of [DCMI\\_Synchronization\\_Mode](#)
- *uint32\_t DCMI\_InitTypeDef::PCKPolarity*  
Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of [DCMI\\_PIXCK\\_Polarity](#)
- *uint32\_t DCMI\_InitTypeDef::VSPolarity*  
Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of [DCMI\\_VSYNC\\_Polarity](#)
- *uint32\_t DCMI\_InitTypeDef::HSPolarity*  
Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of [DCMI\\_HSYNC\\_Polarity](#)

- **`uint32_t DCMI_InitTypeDef::CaptureRate`**  
Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of [DCMI\\_Capture\\_Rate](#)
- **`uint32_t DCMI_InitTypeDef::ExtendedDataMode`**  
Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of [DCMI\\_Extended\\_Data\\_Mode](#)
- **`DCMI_CodesInitTypeDef DCMI_InitTypeDef::SyncroCode`**  
Specifies the code of the frame start delimiter.
- **`uint32_t DCMI_InitTypeDef::JPEGMode`**  
Enable or Disable the JPEG mode This parameter can be a value of [DCMI\\_MODE\\_JPEG](#)
- **`uint32_t DCMI_InitTypeDef::ByteSelectMode`**  
Specifies the data to be captured by the interface This parameter can be a value of [DCMIEx\\_Byte\\_Select\\_Mode](#)
- **`uint32_t DCMI_InitTypeDef::ByteSelectStart`**  
Specifies if the data to be captured by the interface is even or odd This parameter can be a value of [DCMIEx\\_Byte\\_Select\\_Start](#)
- **`uint32_t DCMI_InitTypeDef::LineSelectMode`**  
Specifies the line of data to be captured by the interface This parameter can be a value of [DCMIEx\\_Line\\_Select\\_Mode](#)
- **`uint32_t DCMI_InitTypeDef::LineSelectStart`**  
Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of [DCMIEx\\_Line\\_Select\\_Start](#)

## 18.2 DCMIEx Firmware driver defines

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

### 18.2.1 DCMIEx

DCMIEx

***DCMI Byte Select Mode***

#### DCMI\_BSM\_ALL

Interface captures all received data

#### DCMI\_BSM\_OTHER

Interface captures every other byte from the received data

#### DCMI\_BSM\_ALTERNATE\_4

Interface captures one byte out of four

#### DCMI\_BSM\_ALTERNATE\_2

Interface captures two bytes out of four

***DCMI Byte Select Start***

#### DCMI\_OEBS\_ODD

Interface captures first data from the frame/line start, second one being dropped

#### DCMI\_OEBS\_EVEN

Interface captures second data from the frame/line start, first one being dropped

***DCMI Line Select Mode***

#### DCMI\_LSM\_ALL

Interface captures all received lines

#### DCMI\_LSM\_ALTERNATE\_2

Interface captures one line out of two

***DCMI Line Select Start*****DCMI\_OELS\_ODD**

Interface captures first line from the frame start, second one being dropped

**DCMI\_OELS\_EVEN**

Interface captures second line from the frame start, first one being dropped

## 19 HAL DFSDM Generic Driver

### 19.1 DFSDM Firmware driver registers structures

#### 19.1.1 DFSDM\_Channel\_OutputClockTypeDef

*DFSDM\_Channel\_OutputClockTypeDef* is defined in the `stm32f4xx_hal_dfldm.h`

##### Data Fields

- *FunctionalState Activation*
- *uint32\_t Selection*
- *uint32\_t Divider*

##### Field Documentation

- *FunctionalState DFSDM\_Channel\_OutputClockTypeDef::Activation*  
Output clock enable/disable
- *uint32\_t DFSDM\_Channel\_OutputClockTypeDef::Selection*  
Output clock is system clock or audio clock. This parameter can be a value of [DFSDM\\_Channel\\_OuputClock](#)
- *uint32\_t DFSDM\_Channel\_OutputClockTypeDef::Divider*  
Output clock divider. This parameter must be a number between `Min_Data = 2` and `Max_Data = 256`

#### 19.1.2 DFSDM\_Channel\_InputTypeDef

*DFSDM\_Channel\_InputTypeDef* is defined in the `stm32f4xx_hal_dfldm.h`

##### Data Fields

- *uint32\_t Multiplexer*
- *uint32\_t DataPacking*
- *uint32\_t Pins*

##### Field Documentation

- *uint32\_t DFSDM\_Channel\_InputTypeDef::Multiplexer*  
Input is external serial inputs or internal register. This parameter can be a value of [DFSDM\\_Channel\\_InputMultiplexer](#)
- *uint32\_t DFSDM\_Channel\_InputTypeDef::DataPacking*  
Standard, interleaved or dual mode for internal register. This parameter can be a value of [DFSDM\\_Channel\\_DataPacking](#)
- *uint32\_t DFSDM\_Channel\_InputTypeDef::Pins*  
Input pins are taken from same or following channel. This parameter can be a value of [DFSDM\\_Channel\\_InputPins](#)

#### 19.1.3 DFSDM\_Channel\_SerialInterfaceTypeDef

*DFSDM\_Channel\_SerialInterfaceTypeDef* is defined in the `stm32f4xx_hal_dfldm.h`

##### Data Fields

- *uint32\_t Type*
- *uint32\_t SpiClock*

##### Field Documentation

- *uint32\_t DFSDM\_Channel\_SerialInterfaceTypeDef::Type*  
SPI or Manchester modes. This parameter can be a value of [DFSDM\\_Channel\\_SerialInterfaceType](#)
- *uint32\_t DFSDM\_Channel\_SerialInterfaceTypeDef::SpiClock*  
SPI clock select (external or internal with different sampling point). This parameter can be a value of [DFSDM\\_Channel\\_SpiClock](#)

### 19.1.4 DFSDM\_Channel\_AwdTypeDef

*DFSDM\_Channel\_AwdTypeDef* is defined in the stm32f4xx\_hal\_dfldm.h

#### Data Fields

- *uint32\_t FilterOrder*
- *uint32\_t Oversampling*

#### Field Documentation

- *uint32\_t DFSDM\_Channel\_AwdTypeDef::FilterOrder*  
Analog watchdog Sinc filter order. This parameter can be a value of *DFSDM\_Channel\_AwdFilterOrder*
- *uint32\_t DFSDM\_Channel\_AwdTypeDef::Oversampling*  
Analog watchdog filter oversampling ratio. This parameter must be a number between Min\_Data = 1 and Max\_Data = 32

### 19.1.5 DFSDM\_Channel\_InitTypeDef

*DFSDM\_Channel\_InitTypeDef* is defined in the stm32f4xx\_hal\_dfldm.h

#### Data Fields

- *DFSDM\_Channel\_OutputClockTypeDef OutputClock*
- *DFSDM\_Channel\_InputTypeDef Input*
- *DFSDM\_Channel\_SerialInterfaceTypeDef SerialInterface*
- *DFSDM\_Channel\_AwdTypeDef Awd*
- *int32\_t Offset*
- *uint32\_t RightBitShift*

#### Field Documentation

- *DFSDM\_Channel\_OutputClockTypeDef DFSDM\_Channel\_InitTypeDef::OutputClock*  
DFSDM channel output clock parameters
- *DFSDM\_Channel\_InputTypeDef DFSDM\_Channel\_InitTypeDef::Input*  
DFSDM channel input parameters
- *DFSDM\_Channel\_SerialInterfaceTypeDef DFSDM\_Channel\_InitTypeDef::SerialInterface*  
DFSDM channel serial interface parameters
- *DFSDM\_Channel\_AwdTypeDef DFSDM\_Channel\_InitTypeDef::Awd*  
DFSDM channel analog watchdog parameters
- *int32\_t DFSDM\_Channel\_InitTypeDef::Offset*  
DFSDM channel offset. This parameter must be a number between Min\_Data = -8388608 and Max\_Data = 8388607
- *uint32\_t DFSDM\_Channel\_InitTypeDef::RightBitShift*  
DFSDM channel right bit shift. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F

### 19.1.6 DFSDM\_Channel\_HandleTypeDef

*DFSDM\_Channel\_HandleTypeDef* is defined in the stm32f4xx\_hal\_dfldm.h

#### Data Fields

- *DFSDM\_Channel\_TypeDef \* Instance*
- *DFSDM\_Channel\_InitTypeDef Init*
- *HAL\_DFSDM\_Channel\_StateTypeDef State*

#### Field Documentation

- *DFSDM\_Channel\_TypeDef\* DFSDM\_Channel\_HandleTypeDef::Instance*  
DFSDM channel instance
- *DFSDM\_Channel\_InitTypeDef DFSDM\_Channel\_HandleTypeDef::Init*  
DFSDM channel init parameters
- *HAL\_DFSDM\_Channel\_StateTypeDef DFSDM\_Channel\_HandleTypeDef::State*  
DFSDM channel state

### 19.1.7 DFSDM\_Filter\_RegularParamTypeDef

*DFSDM\_Filter\_RegularParamTypeDef* is defined in the `stm32f4xx_hal_dfldm.h`

#### Data Fields

- *uint32\_t Trigger*
- *FunctionalState FastMode*
- *FunctionalState DmaMode*

#### Field Documentation

- *uint32\_t DFSDM\_Filter\_RegularParamTypeDef::Trigger*  
Trigger used to start regular conversion: software or synchronous. This parameter can be a value of [DFSDM\\_Filter\\_Trigger](#)
- *FunctionalState DFSDM\_Filter\_RegularParamTypeDef::FastMode*  
Enable/disable fast mode for regular conversion
- *FunctionalState DFSDM\_Filter\_RegularParamTypeDef::DmaMode*  
Enable/disable DMA for regular conversion

### 19.1.8 DFSDM\_Filter\_InjectedParamTypeDef

*DFSDM\_Filter\_InjectedParamTypeDef* is defined in the `stm32f4xx_hal_dfldm.h`

#### Data Fields

- *uint32\_t Trigger*
- *FunctionalState ScanMode*
- *FunctionalState DmaMode*
- *uint32\_t ExtTrigger*
- *uint32\_t ExtTriggerEdge*

#### Field Documentation

- *uint32\_t DFSDM\_Filter\_InjectedParamTypeDef::Trigger*  
Trigger used to start injected conversion: software, external or synchronous. This parameter can be a value of [DFSDM\\_Filter\\_Trigger](#)
- *FunctionalState DFSDM\_Filter\_InjectedParamTypeDef::ScanMode*  
Enable/disable scanning mode for injected conversion
- *FunctionalState DFSDM\_Filter\_InjectedParamTypeDef::DmaMode*  
Enable/disable DMA for injected conversion
- *uint32\_t DFSDM\_Filter\_InjectedParamTypeDef::ExtTrigger*  
External trigger. This parameter can be a value of [DFSDM\\_Filter\\_ExtTrigger](#)
- *uint32\_t DFSDM\_Filter\_InjectedParamTypeDef::ExtTriggerEdge*  
External trigger edge: rising, falling or both. This parameter can be a value of [DFSDM\\_Filter\\_ExtTriggerEdge](#)

### 19.1.9 DFSDM\_Filter\_FilterParamTypeDef

*DFSDM\_Filter\_FilterParamTypeDef* is defined in the `stm32f4xx_hal_dfldm.h`

#### Data Fields

- *uint32\_t SincOrder*
- *uint32\_t Oversampling*
- *uint32\_t IntOversampling*

#### Field Documentation

- *uint32\_t DFSDM\_Filter\_FilterParamTypeDef::SincOrder*  
Sinc filter order. This parameter can be a value of [DFSDM\\_Filter\\_SincOrder](#)
- *uint32\_t DFSDM\_Filter\_FilterParamTypeDef::Oversampling*  
Filter oversampling ratio. This parameter must be a number between `Min_Data = 1` and `Max_Data = 1024`

- ***uint32\_t DFSDM\_Filter\_FilterParamTypeDef::IntOversampling***  
Integrator oversampling ratio. This parameter must be a number between Min\_Data = 1 and Max\_Data = 256

### 19.1.10 DFSDM\_Filter\_InitTypeDef

***DFSDM\_Filter\_InitTypeDef*** is defined in the `stm32f4xx_hal_dfldsm.h`

#### Data Fields

- ***DFSDM\_Filter\_RegularParamTypeDef RegularParam***
- ***DFSDM\_Filter\_InjectedParamTypeDef InjectedParam***
- ***DFSDM\_Filter\_FilterParamTypeDef FilterParam***

#### Field Documentation

- ***DFSDM\_Filter\_RegularParamTypeDef DFSDM\_Filter\_InitTypeDef::RegularParam***  
DFSDM regular conversion parameters
- ***DFSDM\_Filter\_InjectedParamTypeDef DFSDM\_Filter\_InitTypeDef::InjectedParam***  
DFSDM injected conversion parameters
- ***DFSDM\_Filter\_FilterParamTypeDef DFSDM\_Filter\_InitTypeDef::FilterParam***  
DFSDM filter parameters

### 19.1.11 DFSDM\_Filter\_HandleTypeDef

***DFSDM\_Filter\_HandleTypeDef*** is defined in the `stm32f4xx_hal_dfldsm.h`

#### Data Fields

- ***DFSDM\_Filter\_TypeDef \* Instance***
- ***DFSDM\_Filter\_InitTypeDef Init***
- ***DMA\_HandleTypeDef \* hdmaReg***
- ***DMA\_HandleTypeDef \* hdmaInj***
- ***uint32\_t RegularContMode***
- ***uint32\_t RegularTrigger***
- ***uint32\_t InjectedTrigger***
- ***uint32\_t ExtTriggerEdge***
- ***FunctionalState InjectedScanMode***
- ***uint32\_t InjectedChannelsNbr***
- ***uint32\_t InjConvRemaining***
- ***HAL\_DFSDM\_Filter\_StateTypeDef State***
- ***uint32\_t ErrorCode***

#### Field Documentation

- ***DFSDM\_Filter\_TypeDef\* DFSDM\_Filter\_HandleTypeDef::Instance***  
DFSDM filter instance
- ***DFSDM\_Filter\_InitTypeDef DFSDM\_Filter\_HandleTypeDef::Init***  
DFSDM filter init parameters
- ***DMA\_HandleTypeDef\* DFSDM\_Filter\_HandleTypeDef::hdmaReg***  
Pointer on DMA handler for regular conversions
- ***DMA\_HandleTypeDef\* DFSDM\_Filter\_HandleTypeDef::hdmaInj***  
Pointer on DMA handler for injected conversions
- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::RegularContMode***  
Regular conversion continuous mode
- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::RegularTrigger***  
Trigger used for regular conversion
- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::InjectedTrigger***  
Trigger used for injected conversion

- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::ExtTriggerEdge***  
Rising, falling or both edges selected
- ***FunctionalState DFSDM\_Filter\_HandleTypeDef::InjectedScanMode***  
Injected scanning mode
- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::InjectedChannelsNbr***  
Number of channels in injected sequence
- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::InjConvRemaining***  
Injected conversions remaining
- ***HAL\_DFSDM\_Filter\_StateTypeDef DFSDM\_Filter\_HandleTypeDef::State***  
DFSDM filter state
- ***uint32\_t DFSDM\_Filter\_HandleTypeDef::ErrorCode***  
DFSDM filter error code

### 19.1.12 DFSDM\_Filter\_AwdParamTypeDef

***DFSDM\_Filter\_AwdParamTypeDef*** is defined in the `stm32f4xx_hal_dfldm.h`

#### Data Fields

- ***uint32\_t DataSource***
- ***uint32\_t Channel***
- ***int32\_t HighThreshold***
- ***int32\_t LowThreshold***
- ***uint32\_t HighBreakSignal***
- ***uint32\_t LowBreakSignal***

#### Field Documentation

- ***uint32\_t DFSDM\_Filter\_AwdParamTypeDef::DataSource***  
Values from digital filter or from channel watchdog filter. This parameter can be a value of [DFSDM\\_Filter\\_AwdDataSource](#)
- ***uint32\_t DFSDM\_Filter\_AwdParamTypeDef::Channel***  
Analog watchdog channel selection. This parameter can be a values combination of [DFSDM\\_Channel\\_Selection](#)
- ***int32\_t DFSDM\_Filter\_AwdParamTypeDef::HighThreshold***  
High threshold for the analog watchdog. This parameter must be a number between `Min_Data = -8388608` and `Max_Data = 8388607`
- ***int32\_t DFSDM\_Filter\_AwdParamTypeDef::LowThreshold***  
Low threshold for the analog watchdog. This parameter must be a number between `Min_Data = -8388608` and `Max_Data = 8388607`
- ***uint32\_t DFSDM\_Filter\_AwdParamTypeDef::HighBreakSignal***  
Break signal assigned to analog watchdog high threshold event. This parameter can be a values combination of [DFSDM\\_BreakSignals](#)
- ***uint32\_t DFSDM\_Filter\_AwdParamTypeDef::LowBreakSignal***  
Break signal assigned to analog watchdog low threshold event. This parameter can be a values combination of [DFSDM\\_BreakSignals](#)

### 19.1.13 DFSDM\_MultiChannelConfigTypeDef

***DFSDM\_MultiChannelConfigTypeDef*** is defined in the `stm32f4xx_hal_dfldm.h`

#### Data Fields

- ***uint32\_t DFSDM1ClockIn***
- ***uint32\_t DFSDM2ClockIn***
- ***uint32\_t DFSDM1ClockOut***
- ***uint32\_t DFSDM2ClockOut***
- ***uint32\_t DFSDM1BitClkDistribution***
- ***uint32\_t DFSDM2BitClkDistribution***



- *uint32\_t DFSDM1DataDistribution*
- *uint32\_t DFSDM2DataDistribution*

#### Field Documentation

- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM1ClockIn*  
Source selection for DFSDM1\_Ckin. This parameter can be a value of [DFSDM\\_1\\_CLOCKIN\\_SELECTION](#)
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM2ClockIn*  
Source selection for DFSDM2\_Ckin. This parameter can be a value of [DFSDM\\_2\\_CLOCKIN\\_SELECTION](#)
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM1ClockOut*  
Source selection for DFSDM1\_Ckout. This parameter can be a value of [DFSDM\\_1\\_CLOCKOUT\\_SELECTION](#)
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM2ClockOut*  
Source selection for DFSDM2\_Ckout. This parameter can be a value of [DFSDM\\_2\\_CLOCKOUT\\_SELECTION](#)
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM1BitClkDistribution*  
Distribution of the DFSDM1 bitstream clock gated by TIM4 OC1 or TIM4 OC2. This parameter can be a value of [DFSDM\\_1\\_BIT\\_STREAM\\_DISTRIBUTION](#)  
**Note:**
  - The DFSDM2 audio gated by TIM4 OC2 can be injected on CKIN0 or CKIN2
  - The DFSDM2 audio gated by TIM4 OC1 can be injected on CKIN1 or CKIN3
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM2BitClkDistribution*  
Distribution of the DFSDM2 bitstream clock gated by TIM3 OC1 or TIM3 OC2 or TIM3 OC3 or TIM3 OC4. This parameter can be a value of [DFSDM\\_2\\_BIT\\_STREAM\\_DISTRIBUTION](#)  
**Note:**
  - The DFSDM2 audio gated by TIM3 OC4 can be injected on CKIN0 or CKIN4
  - The DFSDM2 audio gated by TIM3 OC3 can be injected on CKIN1 or CKIN5
  - The DFSDM2 audio gated by TIM3 OC2 can be injected on CKIN2 or CKIN6
  - The DFSDM2 audio gated by TIM3 OC1 can be injected on CKIN3 or CKIN7
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM1DataDistribution*  
Source selection for DatIn0 and DatIn2 of DFSDM1. This parameter can be a value of [DFSDM\\_1\\_DATA\\_DISTRIBUTION](#)
- *uint32\_t DFSDM\_MultiChannelConfigTypeDef::DFSDM2DataDistribution*  
Source selection for DatIn0, DatIn2, DatIn4 and DatIn6 of DFSDM2. This parameter can be a value of [DFSDM\\_2\\_DATA\\_DISTRIBUTION](#)

## 19.2 DFSDM Firmware driver API description

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

### 19.2.1 How to use this driver

#### Channel initialization

1. User has first to initialize channels (before filters initialization).
2. As prerequisite, fill in the HAL\_DFSDM\_ChannelMspInit() :
  - Enable DFSDMz clock interface with `__HAL_RCC_DFSDMz_CLK_ENABLE()`.
  - Enable the clocks for the DFSDMz GPIOs with `__HAL_RCC_GPIOx_CLK_ENABLE()`.
  - Configure these DFSDMz pins in alternate mode using `HAL_GPIO_Init()`.
  - If interrupt mode is used, enable and configure DFSDMz\_FLT0 global interrupt with `HAL_NVIC_SetPriority()` and `HAL_NVIC_EnableIRQ()`.
3. Configure the output clock, input, serial interface, analog watchdog, offset and data right bit shift parameters for this channel using the `HAL_DFSDM_ChannelInit()` function.

### Channel clock absence detector

1. Start clock absence detector using HAL\_DFSDM\_ChannelCkabStart() or HAL\_DFSDM\_ChannelCkabStart\_IT().
2. In polling mode, use HAL\_DFSDM\_ChannelPollForCkab() to detect the clock absence.
3. In interrupt mode, HAL\_DFSDM\_ChannelCkabCallback() will be called if clock absence is detected.
4. Stop clock absence detector using HAL\_DFSDM\_ChannelCkabStop() or HAL\_DFSDM\_ChannelCkabStop\_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if clock absence detector is stopped for one channel, interrupt will be disabled for all channels.

### Channel short circuit detector

1. Start short circuit detector using HAL\_DFSDM\_ChannelScdStart() or HAL\_DFSDM\_ChannelScdStart\_IT().
2. In polling mode, use HAL\_DFSDM\_ChannelPollForScd() to detect short circuit.
3. In interrupt mode, HAL\_DFSDM\_ChannelScdCallback() will be called if short circuit is detected.
4. Stop short circuit detector using HAL\_DFSDM\_ChannelScdStop() or HAL\_DFSDM\_ChannelScdStop\_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if short circuit detector is stopped for one channel, interrupt will be disabled for all channels.

### Channel analog watchdog value

1. Get analog watchdog filter value of a channel using HAL\_DFSDM\_ChannelGetAwdValue().

### Channel offset value

1. Modify offset value of a channel using HAL\_DFSDM\_ChannelModifyOffset().

### Filter initialization

1. After channel initialization, user has to init filters.
2. As prerequisite, fill in the HAL\_DFSDM\_FilterMspInit() :
  - If interrupt mode is used , enable and configure DFSDMz\_FLTx global interrupt with HAL\_NVIC\_SetPriority() and HAL\_NVIC\_EnableIRQ(). Please note that DFSDMz\_FLT0 global interrupt could be already enabled if interrupt is used for channel.
  - If DMA mode is used, configure DMA with HAL\_DMA\_Init() and link it with DFSDMz filter handle using \_\_HAL\_LINKDMA().
3. Configure the regular conversion, injected conversion and filter parameters for this filter using the HAL\_DFSDM\_FilterInit() function.

### Filter regular channel conversion

1. Select regular channel and enable/disable continuous mode using HAL\_DFSDM\_FilterConfigRegChannel().
2. Start regular conversion using HAL\_DFSDM\_FilterRegularStart(), HAL\_DFSDM\_FilterRegularStart\_IT(), HAL\_DFSDM\_FilterRegularStart\_DMA() or HAL\_DFSDM\_FilterRegularMsbStart\_DMA().
3. In polling mode, use HAL\_DFSDM\_FilterPollForRegConversion() to detect the end of regular conversion.
4. In interrupt mode, HAL\_DFSDM\_FilterRegConvCpltCallback() will be called at the end of regular conversion.
5. Get value of regular conversion and corresponding channel using HAL\_DFSDM\_FilterGetRegularValue().
6. In DMA mode, HAL\_DFSDM\_FilterRegConvHalfCpltCallback() and HAL\_DFSDM\_FilterRegConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL\_DFSDM\_FilterRegConvHalfCpltCallback() will be called only in DMA circular mode.

7. Stop regular conversion using `HAL_DFSDM_FilterRegularStop()`, `HAL_DFSDM_FilterRegularStop_IT()` or `HAL_DFSDM_FilterRegularStop_DMA()`.

#### **Filter injected channels conversion**

1. Select injected channels using `HAL_DFSDM_FilterConfigInjChannel()`.
2. Start injected conversion using `HAL_DFSDM_FilterInjectedStart()`, `HAL_DFSDM_FilterInjectedStart_IT()`, `HAL_DFSDM_FilterInjectedStart_DMA()` or `HAL_DFSDM_FilterInjectedMsbStart_DMA()`.
3. In polling mode, use `HAL_DFSDM_FilterPollForInjConversion()` to detect the end of injected conversion.
4. In interrupt mode, `HAL_DFSDM_FilterInjConvCpltCallback()` will be called at the end of injected conversion.
5. Get value of injected conversion and corresponding channel using `HAL_DFSDM_FilterGetInjectedValue()`.
6. In DMA mode, `HAL_DFSDM_FilterInjConvHalfCpltCallback()` and `HAL_DFSDM_FilterInjConvCpltCallback()` will be called respectively at the half transfer and at the transfer complete. Please note that `HAL_DFSDM_FilterInjConvCpltCallback()` will be called only in DMA circular mode.
7. Stop injected conversion using `HAL_DFSDM_FilterInjectedStop()`, `HAL_DFSDM_FilterInjectedStop_IT()` or `HAL_DFSDM_FilterInjectedStop_DMA()`.

#### **Filter analog watchdog**

1. Start filter analog watchdog using `HAL_DFSDM_FilterAwdStart_IT()`.
2. `HAL_DFSDM_FilterAwdCallback()` will be called if analog watchdog occurs.
3. Stop filter analog watchdog using `HAL_DFSDM_FilterAwdStop_IT()`.

#### **Filter extreme detector**

1. Start filter extreme detector using `HAL_DFSDM_FilterExdStart()`.
2. Get extreme detector maximum value using `HAL_DFSDM_FilterGetExdMaxValue()`.
3. Get extreme detector minimum value using `HAL_DFSDM_FilterGetExdMinValue()`.
4. Start filter extreme detector using `HAL_DFSDM_FilterExdStop()`.

#### **Filter conversion time**

1. Get conversion time value using `HAL_DFSDM_FilterGetConvTimeValue()`.

#### **Callback registration**

The compilation define `USE_HAL_DFSDM_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use functions `HAL_DFSDM_Channel_RegisterCallback()`, `HAL_DFSDM_Filter_RegisterCallback()` or `HAL_DFSDM_Filter_RegisterAwdCallback()` to register a user callback.

Function `HAL_DFSDM_Channel_RegisterCallback()` allows to register following callbacks:

- `CkabCallback` : DFSDM channel clock absence detection callback.
- `ScdCallback` : DFSDM channel short circuit detection callback.
- `MspInitCallback` : DFSDM channel MSP init callback.
- `MspDeInitCallback` : DFSDM channel MSP de-init callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Function `HAL_DFSDM_Filter_RegisterCallback()` allows to register following callbacks:

- `RegConvCpltCallback` : DFSDM filter regular conversion complete callback.
- `RegConvHalfCpltCallback` : DFSDM filter half regular conversion complete callback.
- `InjConvCpltCallback` : DFSDM filter injected conversion complete callback.
- `InjConvHalfCpltCallback` : DFSDM filter half injected conversion complete callback.
- `ErrorCallback` : DFSDM filter error callback.
- `MspInitCallback` : DFSDM filter MSP init callback.
- `MspDeInitCallback` : DFSDM filter MSP de-init callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific DFSDM filter analog watchdog callback use dedicated register callback:

`HAL_DFSDM_Filter_RegisterAwdCallback()`.

Use functions `HAL_DFSDM_Channel_UnRegisterCallback()` or `HAL_DFSDM_Filter_UnRegisterCallback()` to reset a callback to the default weak function.

`HAL_DFSDM_Channel_UnRegisterCallback()` takes as parameters the HAL peripheral handle, and the Callback ID.

This function allows to reset following callbacks:

- `CkabCallback` : DFSDM channel clock absence detection callback.
- `ScdCallback` : DFSDM channel short circuit detection callback.
- `MsplnitCallback` : DFSDM channel MSP init callback.
- `MspDelnitCallback` : DFSDM channel MSP de-init callback.

`HAL_DFSDM_Filter_UnRegisterCallback()` takes as parameters the HAL peripheral handle, and the Callback ID.

This function allows to reset following callbacks:

- `RegConvCpltCallback` : DFSDM filter regular conversion complete callback.
- `RegConvHalfCpltCallback` : DFSDM filter half regular conversion complete callback.
- `InjConvCpltCallback` : DFSDM filter injected conversion complete callback.
- `InjConvHalfCpltCallback` : DFSDM filter half injected conversion complete callback.
- `ErrorCallback` : DFSDM filter error callback.
- `MsplnitCallback` : DFSDM filter MSP init callback.
- `MspDelnitCallback` : DFSDM filter MSP de-init callback.

For specific DFSDM filter analog watchdog callback use dedicated unregister callback:

`HAL_DFSDM_Filter_UnRegisterAwdCallback()`.

By default, after the call of init function and if the state is RESET all callbacks are reset to the corresponding legacy weak functions: examples `HAL_DFSDM_ChannelScdCallback()`, `HAL_DFSDM_FilterErrorCallback()`.

Exception done for `Msplnit` and `MspDelnit` callbacks that are respectively reset to the legacy weak functions in the init and de-init only when these callbacks are null (not registered beforehand). If not, `Msplnit` or `MspDelnit` are not null, the init and de-init keep and use the user `Msplnit/MspDelnit` callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for `Msplnit/MspDelnit` callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) `Msplnit/Delnit` callbacks can be used during the init/de-init. In that case first register the `Msplnit/MspDelnit` user callbacks using `HAL_DFSDM_Channel_RegisterCallback()` or `HAL_DFSDM_Filter_RegisterCallback()` before calling init or de-init function.

When The compilation define `USE_HAL_DFSDM_REGISTER_CALLBACKS` is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

## 19.2.2 Channel initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM channel.
- De-initialize the DFSDM channel.

This section contains the following APIs:

- [\*\*\*HAL\\_DFSDM\\_Channellnit\(\)\*\*\*](#)
- [\*\*\*HAL\\_DFSDM\\_ChannelDelnit\(\)\*\*\*](#)
- [\*\*\*HAL\\_DFSDM\\_ChannelMspInnit\(\)\*\*\*](#)
- [\*\*\*HAL\\_DFSDM\\_ChannelMspDelnit\(\)\*\*\*](#)

## 19.2.3 Channel operation functions

This section provides functions allowing to:

- Manage clock absence detector feature.
- Manage short circuit detector feature.
- Get analog watchdog value.
- Modify offset value.

This section contains the following APIs:

- [\*HAL\\_DFSDM\\_ChannelCkabStart\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelPollForCkab\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelCkabStop\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelCkabStart\\_IT\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelCkabCallback\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelCkabStop\\_IT\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelScdStart\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelPollForScd\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelScdStop\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelScdStart\\_IT\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelScdCallback\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelScdStop\\_IT\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelGetAwdValue\(\)\*](#)
- [\*HAL\\_DFSDM\\_ChannelModifyOffset\(\)\*](#)

#### 19.2.4 Channel state function

This section provides function allowing to:

- Get channel handle state.

This section contains the following APIs:

- [\*HAL\\_DFSDM\\_ChannelGetState\(\)\*](#)

#### 19.2.5 Filter initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM filter.
- De-initialize the DFSDM filter.

This section contains the following APIs:

- [\*HAL\\_DFSDM\\_FilterInit\(\)\*](#)
- [\*HAL\\_DFSDM\\_FilterDeInit\(\)\*](#)
- [\*HAL\\_DFSDM\\_FilterMspInit\(\)\*](#)
- [\*HAL\\_DFSDM\\_FilterMspDeInit\(\)\*](#)

#### 19.2.6 Filter control functions

This section provides functions allowing to:

- Select channel and enable/disable continuous mode for regular conversion.
- Select channels for injected conversion.

This section contains the following APIs:

- [\*HAL\\_DFSDM\\_FilterConfigRegChannel\(\)\*](#)
- [\*HAL\\_DFSDM\\_FilterConfigInjChannel\(\)\*](#)

#### 19.2.7 Filter operation functions

This section provides functions allowing to:

- Start conversion of regular/injected channel.
- Poll for the end of regular/injected conversion.
- Stop conversion of regular/injected channel.
- Start conversion of regular/injected channel and enable interrupt.
- Call the callback functions at the end of regular/injected conversions.
- Stop conversion of regular/injected channel and disable interrupt.
- Start conversion of regular/injected channel and enable DMA transfer.

- Stop conversion of regular/injected channel and disable DMA transfer.
- Start analog watchdog and enable interrupt.
- Call the callback function when analog watchdog occurs.
- Stop analog watchdog and disable interrupt.
- Start extreme detector.
- Stop extreme detector.
- Get result of regular channel conversion.
- Get result of injected channel conversion.
- Get extreme detector maximum and minimum values.
- Get conversion time.
- Handle DFSDM interrupt request.

This section contains the following APIs:

- *HAL\_DFSDM\_FilterRegularStart()*
- *HAL\_DFSDM\_FilterPollForRegConversion()*
- *HAL\_DFSDM\_FilterRegularStop()*
- *HAL\_DFSDM\_FilterRegularStart\_IT()*
- *HAL\_DFSDM\_FilterRegularStop\_IT()*
- *HAL\_DFSDM\_FilterRegularStart\_DMA()*
- *HAL\_DFSDM\_FilterRegularMsbStart\_DMA()*
- *HAL\_DFSDM\_FilterRegularStop\_DMA()*
- *HAL\_DFSDM\_FilterGetRegularValue()*
- *HAL\_DFSDM\_FilterInjectedStart()*
- *HAL\_DFSDM\_FilterPollForInjConversion()*
- *HAL\_DFSDM\_FilterInjectedStop()*
- *HAL\_DFSDM\_FilterInjectedStart\_IT()*
- *HAL\_DFSDM\_FilterInjectedStop\_IT()*
- *HAL\_DFSDM\_FilterInjectedStart\_DMA()*
- *HAL\_DFSDM\_FilterInjectedMsbStart\_DMA()*
- *HAL\_DFSDM\_FilterInjectedStop\_DMA()*
- *HAL\_DFSDM\_FilterGetInjectedValue()*
- *HAL\_DFSDM\_FilterAwdStart\_IT()*
- *HAL\_DFSDM\_FilterAwdStop\_IT()*
- *HAL\_DFSDM\_FilterExdStart()*
- *HAL\_DFSDM\_FilterExdStop()*
- *HAL\_DFSDM\_FilterGetExdMaxValue()*
- *HAL\_DFSDM\_FilterGetExdMinValue()*
- *HAL\_DFSDM\_FilterGetConvTimeValue()*
- *HAL\_DFSDM\_IRQHandler()*
- *HAL\_DFSDM\_FilterRegConvCpltCallback()*
- *HAL\_DFSDM\_FilterRegConvHalfCpltCallback()*
- *HAL\_DFSDM\_FilterInjConvCpltCallback()*
- *HAL\_DFSDM\_FilterInjConvHalfCpltCallback()*
- *HAL\_DFSDM\_FilterAwdCallback()*
- *HAL\_DFSDM\_FilterErrorCallback()*

### 19.2.8 Filter state functions

This section provides functions allowing to:

- Get the DFSDM filter state.
- Get the DFSDM filter error.

This section contains the following APIs:

- [HAL\\_DFSDM\\_FilterGetState\(\)](#)
- [HAL\\_DFSDM\\_FilterGetError\(\)](#)

### 19.2.9 Filter MultiChannel operation functions

This section provides functions allowing to:

- Control the DFSDM Multi channel delay block

This section contains the following APIs:

- [HAL\\_DFSDM\\_BitstreamClock\\_Start\(\)](#)
- [HAL\\_DFSDM\\_BitstreamClock\\_Stop\(\)](#)
- [HAL\\_DFSDM\\_DisableDelayClock\(\)](#)
- [HAL\\_DFSDM\\_EnableDelayClock\(\)](#)
- [HAL\\_DFSDM\\_ClockIn\\_SourceSelection\(\)](#)
- [HAL\\_DFSDM\\_ClockOut\\_SourceSelection\(\)](#)
- [HAL\\_DFSDM\\_DataIn0\\_SourceSelection\(\)](#)
- [HAL\\_DFSDM\\_DataIn2\\_SourceSelection\(\)](#)
- [HAL\\_DFSDM\\_DataIn4\\_SourceSelection\(\)](#)
- [HAL\\_DFSDM\\_DataIn6\\_SourceSelection\(\)](#)
- [HAL\\_DFSDM\\_BitStreamClkDistribution\\_Config\(\)](#)
- [HAL\\_DFSDM\\_ConfigMultiChannelDelay\(\)](#)

### 19.2.10 Detailed description of functions

#### HAL\_DFSDM\_Channellnit

##### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_Channellnit (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

##### Function description

Initialize the DFSDM channel according to the specified parameters in the DFSDM\_ChannellnitTypeDef structure and initialize the associated handle.

##### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

##### Return values

- **HAL**: status.

#### HAL\_DFSDM\_ChannelDelnit

##### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelDelnit (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

##### Function description

De-initialize the DFSDM channel.

##### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

##### Return values

- **HAL**: status.

### HAL\_DFSDM\_ChannelMspInit

#### Function name

**void HAL\_DFSDM\_ChannelMspInit (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

#### Function description

Initialize the DFSDM channel MSP.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

#### Return values

- **None**:

### HAL\_DFSDM\_ChannelMspDeInit

#### Function name

**void HAL\_DFSDM\_ChannelMspDeInit (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

#### Function description

De-initialize the DFSDM channel MSP.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

#### Return values

- **None**:

### HAL\_DFSDM\_ChannelCkabStart

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelCkabStart (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

#### Function description

This function allows to start clock absence detection in polling mode.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

#### Return values

- **HAL**: status

#### Notes

- Same mode has to be used for all channels.
- If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL\_TIMEOUT error.

### HAL\_DFSDM\_ChannelCkabStart\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelCkabStart\_IT (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

#### Function description

This function allows to start clock absence detection in interrupt mode.



**Parameters**

- **hdfsdm\_channel**: DFSDM channel handle.

**Return values**

- **HAL**: status

**Notes**

- Same mode has to be used for all channels.
- If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL\_TIMEOUT error.

**HAL\_DFSDM\_ChannelCkabStop**
**Function name**

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelCkabStop (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

**Function description**

This function allows to stop clock absence detection in polling mode.

**Parameters**

- **hdfsdm\_channel**: DFSDM channel handle.

**Return values**

- **HAL**: status

**HAL\_DFSDM\_ChannelCkabStop\_IT**
**Function name**

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelCkabStop\_IT (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

**Function description**

This function allows to stop clock absence detection in interrupt mode.

**Parameters**

- **hdfsdm\_channel**: DFSDM channel handle.

**Return values**

- **HAL**: status

**Notes**

- Interrupt will be disabled for all channels

**HAL\_DFSDM\_ChannelScdStart**
**Function name**

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelScdStart (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel, uint32\_t Threshold, uint32\_t BreakSignal)**

**Function description**

This function allows to start short circuit detection in polling mode.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.
- **Threshold**: Short circuit detector threshold. This parameter must be a number between Min\_Data = 0 and Max\_Data = 255.
- **BreakSignal**: Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.

### Return values

- **HAL**: status

### Notes

- Same mode has to be used for all channels

#### HAL\_DFSDM\_ChannelScdStart\_IT

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelScdStart\_IT (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel, uint32\_t Threshold, uint32\_t BreakSignal)**

### Function description

This function allows to start short circuit detection in interrupt mode.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.
- **Threshold**: Short circuit detector threshold. This parameter must be a number between Min\_Data = 0 and Max\_Data = 255.
- **BreakSignal**: Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.

### Return values

- **HAL**: status

### Notes

- Same mode has to be used for all channels

#### HAL\_DFSDM\_ChannelScdStop

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelScdStop (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

### Function description

This function allows to stop short circuit detection in polling mode.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

### Return values

- **HAL**: status

#### HAL\_DFSDM\_ChannelScdStop\_IT

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_ChannelScdStop\_IT (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)**

### Function description

This function allows to stop short circuit detection in interrupt mode.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

### Return values

- **HAL**: status

### Notes

- Interrupt will be disabled for all channels

### HAL\_DFSDM\_ChannelGetAwdValue

#### Function name

`int16_t HAL_DFSDM_ChannelGetAwdValue (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)`

#### Function description

This function allows to get channel analog watchdog value.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

### Return values

- **Channel**: analog watchdog value.

### HAL\_DFSDM\_ChannelModifyOffset

#### Function name

`HAL_StatusTypeDef HAL_DFSDM_ChannelModifyOffset (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, int32_t Offset)`

#### Function description

This function allows to modify channel offset value.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.
- **Offset**: DFSDM channel offset. This parameter must be a number between `Min_Data = -8388608` and `Max_Data = 8388607`.

### Return values

- **HAL**: status.

### HAL\_DFSDM\_ChannelPollForCkab

#### Function name

`HAL_StatusTypeDef HAL_DFSDM_ChannelPollForCkab (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Timeout)`

#### Function description

This function allows to poll for the clock absence detection.

### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.
- **Timeout**: Timeout value in milliseconds.

### Return values

- **HAL**: status

### HAL\_DFSDM\_ChannelPollForScd

#### Function name

HAL\_StatusTypeDef HAL\_DFSDM\_ChannelPollForScd (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel, uint32\_t Timeout)

#### Function description

This function allows to poll for the short circuit detection.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.
- **Timeout**: Timeout value in milliseconds.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_ChannelCkabCallback

#### Function name

void HAL\_DFSDM\_ChannelCkabCallback (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)

#### Function description

Clock absence detection callback.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

#### Return values

- **None**:

### HAL\_DFSDM\_ChannelScdCallback

#### Function name

void HAL\_DFSDM\_ChannelScdCallback (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)

#### Function description

Short circuit detection callback.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

#### Return values

- **None**:

### HAL\_DFSDM\_ChannelGetState

#### Function name

HAL\_DFSDM\_Channel\_StateTypeDef HAL\_DFSDM\_ChannelGetState (DFSDM\_Channel\_HandleTypeDef \* hdfsdm\_channel)

#### Function description

This function allows to get the current DFSDM channel handle state.

#### Parameters

- **hdfsdm\_channel**: DFSDM channel handle.

#### Return values

- **DFSDM**: channel state.

### HAL\_DFSDM\_FilterInit

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInit (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

Initialize the DFSDM filter according to the specified parameters in the DFSDM\_FilterInitTypeDef structure and initialize the associated handle.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status.

### HAL\_DFSDM\_FilterDeInit

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterDeInit (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

De-initializes the DFSDM filter.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status.

### HAL\_DFSDM\_FilterMspInit

#### Function name

**void HAL\_DFSDM\_FilterMspInit (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

Initializes the DFSDM filter MSP.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **None**:

### HAL\_DFSDM\_FilterMspDeInit

#### Function name

**void HAL\_DFSDM\_FilterMspDeInit (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

De-initializes the DFSDM filter MSP.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **None**:

### HAL\_DFSDM\_FilterConfigRegChannel

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterConfigRegChannel (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t Channel, uint32\_t ContinuousMode)**

#### Function description

This function allows to select channel and to enable/disable continuous mode for regular conversion.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Channel for regular conversion. This parameter can be a value of DFSDM Channel Selection.
- **ContinuousMode**: Enable/disable continuous mode for regular conversion. This parameter can be a value of DFSDM Continuous Mode.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_FilterConfigInjChannel

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterConfigInjChannel (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t Channel)**

#### Function description

This function allows to select channels for injected conversion.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Channels for injected conversion. This parameter can be a values combination of DFSDM Channel Selection.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_FilterRegularStart

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularStart (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to start regular conversion in polling mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

### HAL\_DFSDM\_FilterRegularStart\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularStart\_IT (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

### Function description

This function allows to start regular conversion in interrupt mode.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

## HAL\_DFSDM\_FilterRegularStart\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularStart\_DMA (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, int32\_t \* pData, uint32\_t Length)**

### Function description

This function allows to start regular conversion in DMA mode.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **pData**: The destination buffer address.
- **Length**: The length of data to be transferred from DFSDM filter to memory.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed regular conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

## HAL\_DFSDM\_FilterRegularMsbStart\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularMsbStart\_DMA (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, int16\_t \* pData, uint32\_t Length)**

### Function description

This function allows to start regular conversion in DMA mode and to get only the 16 most significant bits of conversion.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **pData**: The destination buffer address.
- **Length**: The length of data to be transferred from DFSDM filter to memory.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of regular conversion.

### HAL\_DFSDM\_FilterRegularStop

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularStop (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop regular conversion in polling mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if regular conversion is ongoing.

### HAL\_DFSDM\_FilterRegularStop\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularStop\_IT (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop regular conversion in interrupt mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if regular conversion is ongoing.

### HAL\_DFSDM\_FilterRegularStop\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterRegularStop\_DMA (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop regular conversion in DMA mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if regular conversion is ongoing.

### HAL\_DFSDM\_FilterInjectedStart

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedStart (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**



### Function description

This function allows to start injected conversion in polling mode.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

### HAL\_DFSDM\_FilterInjectedStart\_IT

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedStart\_IT (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

### Function description

This function allows to start injected conversion in interrupt mode.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

### HAL\_DFSDM\_FilterInjectedStart\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedStart\_DMA (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, int32\_t \* pData, uint32\_t Length)**

### Function description

This function allows to start injected conversion in DMA mode.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **pData**: The destination buffer address.
- **Length**: The length of data to be transferred from DFSDM filter to memory.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed injected conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

### HAL\_DFSDM\_FilterInjectedMsbStart\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedMsbStart\_DMA (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, int16\_t \* pData, uint32\_t Length)**

### Function description

This function allows to start injected conversion in DMA mode and to get only the 16 most significant bits of conversion.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **pData**: The destination buffer address.
- **Length**: The length of data to be transferred from DFSDM filter to memory.

### Return values

- **HAL**: status

### Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of injected conversion.

### **HAL\_DFSDM\_FilterInjectedStop**

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedStop (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop injected conversion in polling mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if injected conversion is ongoing.

### **HAL\_DFSDM\_FilterInjectedStop\_IT**

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedStop\_IT (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop injected conversion in interrupt mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if injected conversion is ongoing.

### **HAL\_DFSDM\_FilterInjectedStop\_DMA**

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterInjectedStop\_DMA (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop injected conversion in DMA mode.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **HAL**: status

### Notes

- This function should be called only if injected conversion is ongoing.

### HAL\_DFSDM\_FilterAwdStart\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterAwdStart\_IT (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, DFSDM\_Filter\_AwdParamTypeDef \* awdParam)**

#### Function description

This function allows to start filter analog watchdog in interrupt mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **awdParam**: DFSDM filter analog watchdog parameters.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_FilterAwdStop\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterAwdStop\_IT (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

This function allows to stop filter analog watchdog in interrupt mode.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_FilterExdStart

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterExdStart (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t Channel)**

#### Function description

This function allows to start extreme detector feature.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Channels where extreme detector is enabled. This parameter can be a values combination of DFSDM Channel Selection.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_FilterExdStop

#### Function name

HAL\_StatusTypeDef HAL\_DFSDM\_FilterExdStop (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)

#### Function description

This function allows to stop extreme detector feature.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **HAL**: status

### HAL\_DFSDM\_FilterGetRegularValue

#### Function name

int32\_t HAL\_DFSDM\_FilterGetRegularValue (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t \* Channel)

#### Function description

This function allows to get regular conversion value.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Corresponding channel of regular conversion.

#### Return values

- **Regular**: conversion value

### HAL\_DFSDM\_FilterGetInjectedValue

#### Function name

int32\_t HAL\_DFSDM\_FilterGetInjectedValue (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t \* Channel)

#### Function description

This function allows to get injected conversion value.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Corresponding channel of injected conversion.

#### Return values

- **Injected**: conversion value

### HAL\_DFSDM\_FilterGetExdMaxValue

#### Function name

int32\_t HAL\_DFSDM\_FilterGetExdMaxValue (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t \* Channel)

#### Function description

This function allows to get extreme detector maximum value.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Corresponding channel.

### Return values

- **Extreme**: detector maximum value This value is between Min\_Data = -8388608 and Max\_Data = 8388607.

### HAL\_DFSDM\_FilterGetExdMinValue

#### Function name

`int32_t HAL_DFSDM_FilterGetExdMinValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)`

#### Function description

This function allows to get extreme detector minimum value.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Corresponding channel.

### Return values

- **Extreme**: detector minimum value This value is between Min\_Data = -8388608 and Max\_Data = 8388607.

### HAL\_DFSDM\_FilterGetConvTimeValue

#### Function name

`uint32_t HAL_DFSDM_FilterGetConvTimeValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)`

#### Function description

This function allows to get conversion time value.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **Conversion**: time value

### Notes

- To get time in second, this value has to be divided by DFSDM clock frequency.

### HAL\_DFSDM\_IRQHandler

#### Function name

`void HAL_DFSDM_IRQHandler (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)`

#### Function description

This function handles the DFSDM interrupts.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **None**:

### HAL\_DFSDM\_FilterPollForRegConversion

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterPollForRegConversion (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t Timeout)**

#### Function description

This function allows to poll for the end of regular conversion.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Timeout**: Timeout value in milliseconds.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if regular conversion is ongoing.

### HAL\_DFSDM\_FilterPollForInjConversion

#### Function name

**HAL\_StatusTypeDef HAL\_DFSDM\_FilterPollForInjConversion (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t Timeout)**

#### Function description

This function allows to poll for the end of injected conversion.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Timeout**: Timeout value in milliseconds.

#### Return values

- **HAL**: status

#### Notes

- This function should be called only if injected conversion is ongoing.

### HAL\_DFSDM\_FilterRegConvCpltCallback

#### Function name

**void HAL\_DFSDM\_FilterRegConvCpltCallback (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

Regular conversion complete callback.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **None**:

#### Notes

- In interrupt mode, user has to read conversion value in this function using HAL\_DFSDM\_FilterGetRegularValue.

### HAL\_DFSDM\_FilterRegConvHalfCpltCallback

#### Function name

**void HAL\_DFSDM\_FilterRegConvHalfCpltCallback (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

Half regular conversion complete callback.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **None**:

### HAL\_DFSDM\_FilterInjConvCpltCallback

#### Function name

**void HAL\_DFSDM\_FilterInjConvCpltCallback (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

Injected conversion complete callback.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **None**:

#### Notes

- In interrupt mode, user has to read conversion value in this function using HAL\_DFSDM\_FilterGetInjectedValue.

### HAL\_DFSDM\_FilterInjConvHalfCpltCallback

#### Function name

**void HAL\_DFSDM\_FilterInjConvHalfCpltCallback (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter)**

#### Function description

Half injected conversion complete callback.

#### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

#### Return values

- **None**:

### HAL\_DFSDM\_FilterAwdCallback

#### Function name

**void HAL\_DFSDM\_FilterAwdCallback (DFSDM\_Filter\_HandleTypeDef \* hdfsdm\_filter, uint32\_t Channel, uint32\_t Threshold)**

#### Function description

Filter analog watchdog callback.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.
- **Channel**: Corresponding channel.
- **Threshold**: Low or high threshold has been reached.

### Return values

- **None**:

### HAL\_DFSDM\_FilterErrorCallback

### Function name

```
void HAL_DFSDM_FilterErrorCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)
```

### Function description

Error callback.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **None**:

### HAL\_DFSDM\_FilterGetState

### Function name

```
HAL_DFSDM_Filter_StateTypeDef HAL_DFSDM_FilterGetState (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)
```

### Function description

This function allows to get the current DFSDM filter handle state.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **DFSDM**: filter state.

### HAL\_DFSDM\_FilterGetError

### Function name

```
uint32_t HAL_DFSDM_FilterGetError (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)
```

### Function description

This function allows to get the current DFSDM filter error.

### Parameters

- **hdfsdm\_filter**: DFSDM filter handle.

### Return values

- **DFSDM**: filter error code.

### HAL\_DFSDM\_ConfigMultiChannelDelay

### Function name

```
void HAL_DFSDM_ConfigMultiChannelDelay (DFSDM_MultiChannelConfigTypeDef * mchdlystruct)
```



### Function description

Configure multi channel delay block: Use DFSDM2 audio clock source as input clock for DFSDM1 and DFSDM2 filters to Synchronize DFSDMx filters.

### Parameters

- **mchdlystruct:** Structure of multi channel configuration

### Return values

- **None:**

### Notes

- The SYSCFG clock marco `__HAL_RCC_SYSCFG_CLK_ENABLE()` must be called before `HAL_DFSDM_ConfigMultiChannelDelay()`
- The `HAL_DFSDM_ConfigMultiChannelDelay()` function clears the SYSCFG-MCHDLYCR register before setting the new configuration.

### **HAL\_DFSDM\_BitstreamClock\_Start**

#### Function name

**void HAL\_DFSDM\_BitstreamClock\_Start (void )**

#### Function description

Select the DFSDM2 as clock source for the bitstream clock.

#### Notes

- The SYSCFG clock marco `__HAL_RCC_SYSCFG_CLK_ENABLE()` must be called before `HAL_DFSDM_BitstreamClock_Start()`

### **HAL\_DFSDM\_BitstreamClock\_Stop**

#### Function name

**void HAL\_DFSDM\_BitstreamClock\_Stop (void )**

#### Function description

Stop the DFSDM2 as clock source for the bitstream clock.

#### Return values

- **None:**

#### Notes

- The SYSCFG clock marco `__HAL_RCC_SYSCFG_CLK_ENABLE()` must be called before `HAL_DFSDM_BitstreamClock_Stop()`

### **HAL\_DFSDM\_DisableDelayClock**

#### Function name

**void HAL\_DFSDM\_DisableDelayClock (uint32\_t MCHDLY)**

#### Function description

Disable Delay Clock for DFSDM1/2.

#### Parameters

- **MCHDLY:** `HAL_MCHDLY_CLOCK_DFSDM2`. `HAL_MCHDLY_CLOCK_DFSDM1`.

#### Return values

- **None:**

**Notes**

- The SYSCFG clock marco `__HAL_RCC_SYSCFG_CLK_ENABLE()` must be called before `HAL_DFSDM_DisableDelayClock()`

**HAL\_DFSDM\_EnableDelayClock**
**Function name**

```
void HAL_DFSDM_EnableDelayClock (uint32_t MCHDLY)
```

**Function description**

Enable Delay Clock for DFSDM1/2.

**Parameters**

- **MCHDLY:** `HAL_MCHDLY_CLOCK_DFSDM2`. `HAL_MCHDLY_CLOCK_DFSDM1`.

**Return values**

- **None:**

**Notes**

- The SYSCFG clock marco `__HAL_RCC_SYSCFG_CLK_ENABLE()` must be called before `HAL_DFSDM_EnableDelayClock()`

**HAL\_DFSDM\_ClockIn\_SourceSelection**
**Function name**

```
void HAL_DFSDM_ClockIn_SourceSelection (uint32_t source)
```

**Function description**

Select the source for CKin signals for DFSDM1/2.

**Parameters**

- **source:** `DFSDM2_CKIN_PAD`. `DFSDM2_CKIN_DM`. `DFSDM1_CKIN_PAD`. `DFSDM1_CKIN_DM`.

**Return values**

- **None:**

**HAL\_DFSDM\_ClockOut\_SourceSelection**
**Function name**

```
void HAL_DFSDM_ClockOut_SourceSelection (uint32_t source)
```

**Function description**

Select the source for CKOut signals for DFSDM1/2.

**Parameters**

- **source:** `DFSDM2_CKOUT_DFSDM2`. `DFSDM2_CKOUT_M27`. `DFSDM1_CKOUT_DFSDM1`. `DFSDM1_CKOUT_M27`.

**Return values**

- **None:**

**HAL\_DFSDM\_DataIn0\_SourceSelection**
**Function name**

```
void HAL_DFSDM_DataIn0_SourceSelection (uint32_t source)
```

**Function description**

Select the source for DataIn0 signals for DFSDM1/2.

**Parameters**

- **source:** DATAIN0\_DFSDM2\_PAD. DATAIN0\_DFSDM2\_DATAIN1. DATAIN0\_DFSDM1\_PAD. DATAIN0\_DFSDM1\_DATAIN1.

**Return values**

- **None:**

**HAL\_DFSDM\_DataIn2\_SourceSelection**

**Function name**

**void HAL\_DFSDM\_DataIn2\_SourceSelection (uint32\_t source)**

**Function description**

Select the source for DataIn2 signals for DFSDM1/2.

**Parameters**

- **source:** DATAIN2\_DFSDM2\_PAD. DATAIN2\_DFSDM2\_DATAIN3. DATAIN2\_DFSDM1\_PAD. DATAIN2\_DFSDM1\_DATAIN3.

**Return values**

- **None:**

**HAL\_DFSDM\_DataIn4\_SourceSelection**

**Function name**

**void HAL\_DFSDM\_DataIn4\_SourceSelection (uint32\_t source)**

**Function description**

Select the source for DataIn4 signals for DFSDM2.

**Parameters**

- **source:** DATAIN4\_DFSDM2\_PAD. DATAIN4\_DFSDM2\_DATAIN5

**Return values**

- **None:**

**HAL\_DFSDM\_DataIn6\_SourceSelection**

**Function name**

**void HAL\_DFSDM\_DataIn6\_SourceSelection (uint32\_t source)**

**Function description**

Select the source for DataIn6 signals for DFSDM2.

**Parameters**

- **source:** DATAIN6\_DFSDM2\_PAD. DATAIN6\_DFSDM2\_DATAIN7.

**Return values**

- **None:**

**HAL\_DFSDM\_BitStreamClkDistribution\_Config**

**Function name**

**void HAL\_DFSDM\_BitStreamClkDistribution\_Config (uint32\_t source)**

**Function description**

Configure the distribution of the bitstream clock gated from TIM4\_OC for DFSDM1 or TIM3\_OC for DFSDM2.

**Parameters**

- **source:** DFSDM1\_CLKIN0\_TIM4OC2 DFSDM1\_CLKIN2\_TIM4OC2 DFSDM1\_CLKIN1\_TIM4OC1 DFSDM1\_CLKIN3\_TIM4OC1 DFSDM2\_CLKIN0\_TIM3OC4 DFSDM2\_CLKIN4\_TIM3OC4 DFSDM2\_CLKIN1\_TIM3OC3 DFSDM2\_CLKIN5\_TIM3OC3 DFSDM2\_CLKIN2\_TIM3OC2 DFSDM2\_CLKIN6\_TIM3OC2 DFSDM2\_CLKIN3\_TIM3OC1 DFSDM2\_CLKIN7\_TIM3OC1

**Return values**

- **None:**

## 19.3 DFSDM Firmware driver defines

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

### 19.3.1 DFSDM

DFSDM

#### *DFSDM1 Bit Stream Distribution*

DFSDM1\_T4\_OC2\_BITSTREAM\_CKIN0

DFSDM1\_T4\_OC2\_BITSTREAM\_CKIN2

DFSDM1\_T4\_OC1\_BITSTREAM\_CKIN3

DFSDM1\_T4\_OC1\_BITSTREAM\_CKIN1

#### *DFSDM1 ClockIn Selection*

DFSDM1\_CKIN\_DFSDM2\_CKOUT

DFSDM1\_CKIN\_PAD

#### *DFSDM1 ClockOut Selection*

DFSDM1\_CKOUT\_DFSDM2\_CKOUT

DFSDM1\_CKOUT\_DFSDM1

#### *DFSDM1 Data Distribution*

DFSDM1\_DATIN0\_TO\_DATIN0\_PAD

DFSDM1\_DATIN0\_TO\_DATIN1\_PAD

DFSDM1\_DATIN2\_TO\_DATIN2\_PAD

DFSDM1\_DATIN2\_TO\_DATIN3\_PAD

#### *DFSDM12 Bit Stream Distribution*

DFSDM2\_T3\_OC4\_BITSTREAM\_CKIN0

DFSDM2\_T3\_OC4\_BITSTREAM\_CKIN4

DFSDM2\_T3\_OC3\_BITSTREAM\_CKIN5

DFSDM2\_T3\_OC3\_BITSTREAM\_CKIN1

DFSDM2\_T3\_OC2\_BITSTREAM\_CKIN6

DFSDM2\_T3\_OC2\_BITSTREAM\_CKIN2

DFSDM2\_T3\_OC1\_BITSTREAM\_CKIN3

DFSDM2\_T3\_OC1\_BITSTREAM\_CKIN7

***DFSDM2 ClockIn Selection***

DFSDM2\_CKIN\_DFSDM2\_CKOUT

DFSDM2\_CKIN\_PAD

***DFSDM2 ClockOut Selection***

DFSDM2\_CKOUT\_DFSDM2\_CKOUT

DFSDM2\_CKOUT\_DFSDM2

***DFSDM2 Data Distribution***

DFSDM2\_DATIN0\_TO\_DATIN0\_PAD

DFSDM2\_DATIN0\_TO\_DATIN1\_PAD

DFSDM2\_DATIN2\_TO\_DATIN2\_PAD

DFSDM2\_DATIN2\_TO\_DATIN3\_PAD

DFSDM2\_DATIN4\_TO\_DATIN4\_PAD

DFSDM2\_DATIN4\_TO\_DATIN5\_PAD

DFSDM2\_DATIN6\_TO\_DATIN6\_PAD

DFSDM2\_DATIN6\_TO\_DATIN7\_PAD

***DFSDM analog watchdog threshold***

DFSDM\_AWD\_HIGH\_THRESHOLD

Analog watchdog high threshold

DFSDM\_AWD\_LOW\_THRESHOLD

Analog watchdog low threshold

***DFSDM break signals***

DFSDM\_NO\_BREAK\_SIGNAL

No break signal

DFSDM\_BREAK\_SIGNAL\_0

Break signal 0

DFSDM\_BREAK\_SIGNAL\_1

Break signal 1

DFSDM\_BREAK\_SIGNAL\_2

Break signal 2

DFSDM\_BREAK\_SIGNAL\_3

Break signal 3

***DFSDM channel analog watchdog filter order***

**DFSDM\_CHANNEL\_FASTSINC\_ORDER**

FastSinc filter type

**DFSDM\_CHANNEL\_SINC1\_ORDER**

Sinc 1 filter type

**DFSDM\_CHANNEL\_SINC2\_ORDER**

Sinc 2 filter type

**DFSDM\_CHANNEL\_SINC3\_ORDER**

Sinc 3 filter type

***DFSDM channel input data packing*****DFSDM\_CHANNEL\_STANDARD\_MODE**

Standard data packing mode

**DFSDM\_CHANNEL\_INTERLEAVED\_MODE**

Interleaved data packing mode

**DFSDM\_CHANNEL\_DUAL\_MODE**

Dual data packing mode

***DFSDM channel input multiplexer*****DFSDM\_CHANNEL\_EXTERNAL\_INPUTS**

Data are taken from external inputs

**DFSDM\_CHANNEL\_INTERNAL\_REGISTER**

Data are taken from internal register

***DFSDM channel input pins*****DFSDM\_CHANNEL\_SAME\_CHANNEL\_PINS**

Input from pins on same channel

**DFSDM\_CHANNEL\_FOLLOWING\_CHANNEL\_PINS**

Input from pins on following channel

***DFSDM channel output clock selection*****DFSDM\_CHANNEL\_OUTPUT\_CLOCK\_SYSTEM**

Source for output clock is system clock

**DFSDM\_CHANNEL\_OUTPUT\_CLOCK\_AUDIO**

Source for output clock is audio clock

***DFSDM Channel Selection*****DFSDM\_CHANNEL\_0****DFSDM\_CHANNEL\_1****DFSDM\_CHANNEL\_2****DFSDM\_CHANNEL\_3****DFSDM\_CHANNEL\_4****DFSDM\_CHANNEL\_5****DFSDM\_CHANNEL\_6**

**DFSDM\_CHANNEL\_7*****DFSDM channel serial interface type*****DFSDM\_CHANNEL\_SPI\_RISING**

SPI with rising edge

**DFSDM\_CHANNEL\_SPI\_FALLING**

SPI with falling edge

**DFSDM\_CHANNEL\_MANCHESTER\_RISING**

Manchester with rising edge

**DFSDM\_CHANNEL\_MANCHESTER\_FALLING**

Manchester with falling edge

***DFSDM channel SPI clock selection*****DFSDM\_CHANNEL\_SPI\_CLOCK\_EXTERNAL**

External SPI clock

**DFSDM\_CHANNEL\_SPI\_CLOCK\_INTERNAL**

Internal SPI clock

**DFSDM\_CHANNEL\_SPI\_CLOCK\_INTERNAL\_DIV2\_FALLING**

Internal SPI clock divided by 2, falling edge

**DFSDM\_CHANNEL\_SPI\_CLOCK\_INTERNAL\_DIV2\_RISING**

Internal SPI clock divided by 2, rising edge

***DFSDM Clock In Source Selection*****HAL\_DFSDM2\_CKIN\_PAD****HAL\_DFSDM2\_CKIN\_DM****HAL\_DFSDM1\_CKIN\_PAD****HAL\_DFSDM1\_CKIN\_DM*****DFSDM Clock Source Selection*****HAL\_DFSDM2\_CKOUT\_DFSDM2****HAL\_DFSDM2\_CKOUT\_M27****HAL\_DFSDM1\_CKOUT\_DFSDM1****HAL\_DFSDM1\_CKOUT\_M27*****DFSDM Continuous Mode*****DFSDM\_CONTINUOUS\_CONV\_OFF**

Conversion are not continuous

**DFSDM\_CONTINUOUS\_CONV\_ON**

Conversion are continuous

***DFSDM Source Selection For DATAIN0*****HAL\_DATAIN0\_DFSDM2\_PAD**

HAL\_DATAIN0\_DFSDM2\_DATAIN1

HAL\_DATAIN0\_DFSDM1\_PAD

HAL\_DATAIN0\_DFSDM1\_DATAIN1

*DFSDM Source Selection For DATAIN2*

HAL\_DATAIN2\_DFSDM2\_PAD

HAL\_DATAIN2\_DFSDM2\_DATAIN3

HAL\_DATAIN2\_DFSDM1\_PAD

HAL\_DATAIN2\_DFSDM1\_DATAIN3

*DFSDM Source Selection For DATAIN4*

HAL\_DATAIN4\_DFSDM2\_PAD

HAL\_DATAIN4\_DFSDM2\_DATAIN5

*DFSDM Source Selection For DATAIN6*

HAL\_DATAIN6\_DFSDM2\_PAD

HAL\_DATAIN6\_DFSDM2\_DATAIN7

*DFSDM Exported Macros*

**\_\_HAL\_DFSDM\_CHANNEL\_RESET\_HANDLE\_STATE**

**Description:**

- Reset DFSDM channel handle state.

**Parameters:**

- `__HANDLE__`: DFSDM channel handle.

**Return value:**

- None

**\_\_HAL\_DFSDM\_FILTER\_RESET\_HANDLE\_STATE**

**Description:**

- Reset DFSDM filter handle state.

**Parameters:**

- `__HANDLE__`: DFSDM filter handle.

**Return value:**

- None

*DFSDM filter analog watchdog data source*

**DFSDM\_FILTER\_AWD\_FILTER\_DATA**

From digital filter

**DFSDM\_FILTER\_AWD\_CHANNEL\_DATA**

From analog watchdog channel

*DFSDM filter error code*

**DFSDM\_FILTER\_ERROR\_NONE**

No error



**DFSDM\_FILTER\_ERROR\_REGULAR\_OVERRUN**

Overrun occurs during regular conversion

**DFSDM\_FILTER\_ERROR\_INJECTED\_OVERRUN**

Overrun occurs during injected conversion

**DFSDM\_FILTER\_ERROR\_DMA**

DMA error occurs

***DFSDM filter external trigger***

**DFSDM\_FILTER\_EXT\_TRIG\_TIM1\_TRGO**

For All DFSDM1/2 filters

**DFSDM\_FILTER\_EXT\_TRIG\_TIM3\_TRGO**

For All DFSDM1/2 filters

**DFSDM\_FILTER\_EXT\_TRIG\_TIM8\_TRGO**

For All DFSDM1/2 filters

**DFSDM\_FILTER\_EXT\_TRIG\_TIM10\_OC1**

For DFSDM1 filter 0 and 1 and DFSDM2 filter 0, 1 and 2

**DFSDM\_FILTER\_EXT\_TRIG\_TIM2\_TRGO**

For DFSDM2 filter 3

**DFSDM\_FILTER\_EXT\_TRIG\_TIM4\_TRGO**

For DFSDM1 filter 0 and 1 and DFSDM2 filter 0, 1 and 2

**DFSDM\_FILTER\_EXT\_TRIG\_TIM11\_OC1**

For DFSDM2 filter 3

**DFSDM\_FILTER\_EXT\_TRIG\_TIM6\_TRGO**

For DFSDM1 filter 0 and 1 and DFSDM2 filter 0 and 1

**DFSDM\_FILTER\_EXT\_TRIG\_TIM7\_TRGO**

For DFSDM2 filter 2 and 3

**DFSDM\_FILTER\_EXT\_TRIG\_EXTI11**

For All DFSDM1/2 filters

**DFSDM\_FILTER\_EXT\_TRIG\_EXTI15**

For All DFSDM1/2 filters

***DFSDM filter external trigger edge***

**DFSDM\_FILTER\_EXT\_TRIG\_RISING\_EDGE**

External rising edge

**DFSDM\_FILTER\_EXT\_TRIG\_FALLING\_EDGE**

External falling edge

**DFSDM\_FILTER\_EXT\_TRIG\_BOTH\_EDGES**

External rising and falling edges

***DFSDM filter sinc order***

**DFSDM\_FILTER\_FASTSINC\_ORDER**

FastSinc filter type

**DFSDM\_FILTER\_SINC1\_ORDER**

Sinc 1 filter type

**DFSDM\_FILTER\_SINC2\_ORDER**

Sinc 2 filter type

**DFSDM\_FILTER\_SINC3\_ORDER**

Sinc 3 filter type

**DFSDM\_FILTER\_SINC4\_ORDER**

Sinc 4 filter type

**DFSDM\_FILTER\_SINC5\_ORDER**

Sinc 5 filter type

***DFSDM filter conversion trigger***

**DFSDM\_FILTER\_SW\_TRIGGER**

Software trigger

**DFSDM\_FILTER\_SYNC\_TRIGGER**

Synchronous with DFSDM\_FLT0

**DFSDM\_FILTER\_EXT\_TRIGGER**

External trigger (only for injected conversion)

## 20 HAL DMA2D Generic Driver

### 20.1 DMA2D Firmware driver registers structures

#### 20.1.1 DMA2D\_CLUTCfgTypeDef

*DMA2D\_CLUTCfgTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

##### Data Fields

- *uint32\_t* \*pCLUT
- *uint32\_t* CLUTColorMode
- *uint32\_t* Size

##### Field Documentation

- *uint32\_t*\* *DMA2D\_CLUTCfgTypeDef::pCLUT*  
Configures the DMA2D CLUT memory address.
- *uint32\_t* *DMA2D\_CLUTCfgTypeDef::CLUTColorMode*  
Configures the DMA2D CLUT color mode. This parameter can be one value of [DMA2D\\_CLUT\\_CM](#).
- *uint32\_t* *DMA2D\_CLUTCfgTypeDef::Size*  
Configures the DMA2D CLUT size. This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF`.

#### 20.1.2 DMA2D\_InitTypeDef

*DMA2D\_InitTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

##### Data Fields

- *uint32\_t* Mode
- *uint32\_t* ColorMode
- *uint32\_t* OutputOffset

##### Field Documentation

- *uint32\_t* *DMA2D\_InitTypeDef::Mode*  
Configures the DMA2D transfer mode. This parameter can be one value of [DMA2D\\_Mode](#).
- *uint32\_t* *DMA2D\_InitTypeDef::ColorMode*  
Configures the color format of the output image. This parameter can be one value of [DMA2D\\_Output\\_Color\\_Mode](#).
- *uint32\_t* *DMA2D\_InitTypeDef::OutputOffset*  
Specifies the Offset value. This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0x3FFF`.

#### 20.1.3 DMA2D\_LayerCfgTypeDef

*DMA2D\_LayerCfgTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

##### Data Fields

- *uint32\_t* InputOffset
- *uint32\_t* InputColorMode
- *uint32\_t* AlphaMode
- *uint32\_t* InputAlpha

##### Field Documentation

- *uint32\_t* *DMA2D\_LayerCfgTypeDef::InputOffset*  
Configures the DMA2D foreground or background offset. This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0x3FFF`.
- *uint32\_t* *DMA2D\_LayerCfgTypeDef::InputColorMode*  
Configures the DMA2D foreground or background color mode. This parameter can be one value of [DMA2D\\_Input\\_Color\\_Mode](#).

- **`uint32_t DMA2D_LayerCfgTypeDef::AlphaMode`**  
Configures the DMA2D foreground or background alpha mode. This parameter can be one value of [DMA2D\\_Alpha\\_Mode](#).
- **`uint32_t DMA2D_LayerCfgTypeDef::InputAlpha`**  
Specifies the DMA2D foreground or background alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` except for the color modes detailed below.  
**Note:**
  - In case of A8 or A4 color mode (ARGB), this parameter must be a number between `Min_Data = 0x00000000` and `Max_Data = 0xFFFFFFFF` where
    - `InputAlpha[24:31]` is the alpha value `ALPHA[0:7]`
    - `InputAlpha[16:23]` is the red value `RED[0:7]`
    - `InputAlpha[8:15]` is the green value `GREEN[0:7]`
    - `InputAlpha[0:7]` is the blue value `BLUE[0:7]`.

#### 20.1.4 `__DMA2D_HandleTypeDef`

`__DMA2D_HandleTypeDef` is defined in the `stm32f4xx_hal_dma2d.h`

##### Data Fields

- **`DMA2D_TypeDef * Instance`**
- **`DMA2D_InitTypeDef Init`**
- **`void(* XferCpltCallback`**
- **`void(* XferErrorCallback`**
- **`DMA2D_LayerCfgTypeDef LayerCfg`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_DMA2D_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

##### Field Documentation

- **`DMA2D_TypeDef* __DMA2D_HandleTypeDef::Instance`**  
DMA2D register base address.
- **`DMA2D_InitTypeDef __DMA2D_HandleTypeDef::Init`**  
DMA2D communication parameters.
- **`void(* __DMA2D_HandleTypeDef::XferCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)`**  
DMA2D transfer complete callback.
- **`void(* __DMA2D_HandleTypeDef::XferErrorCallback)(struct __DMA2D_HandleTypeDef *hdma2d)`**  
DMA2D transfer error callback.
- **`DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDef::LayerCfg[MAX_DMA2D_LAYER]`**  
DMA2D Layers parameters
- **`HAL_LockTypeDef __DMA2D_HandleTypeDef::Lock`**  
DMA2D lock.
- **`__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDef::State`**  
DMA2D transfer state.
- **`__IO uint32_t __DMA2D_HandleTypeDef::ErrorCode`**  
DMA2D error code.

## 20.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

### 20.2.1 How to use this driver

1. Program the required configuration through the following parameters: the transfer mode, the output color mode and the output offset using `HAL_DMA2D_Init()` function.

2. Program the required configuration through the following parameters: the input color mode, the input color, the input alpha value, the alpha mode, the red/blue swap mode, the inverted alpha mode and the input offset using HAL\_DMA2D\_ConfigLayer() function for foreground or/and background layer.

#### **Polling mode IO operation**

1. Configure pdata parameter (explained hereafter), destination and data length and enable the transfer using HAL\_DMA2D\_Start().
2. Wait for end of transfer using HAL\_DMA2D\_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

#### **Interrupt mode IO operation**

1. Configure pdata parameter, destination and data length and enable the transfer using HAL\_DMA2D\_Start\_IT().
2. Use HAL\_DMA2D\_IRQHandler() called under DMA2D\_IRQHandler() interrupt subroutine.
3. At the end of data transfer HAL\_DMA2D\_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback (member of DMA2D handle structure).
4. In case of error, the HAL\_DMA2D\_IRQHandler() function calls the callback XferErrorCallback.

*Note:* In Register-to-Memory transfer mode, pdata parameter is the register color, in Memory-to-memory or Memory-to-Memory with pixel format conversion pdata is the source address.

*Note:* Configure the foreground source address, the background source address, the destination and data length then Enable the transfer using HAL\_DMA2D\_BlendingStart() in polling mode and HAL\_DMA2D\_BlendingStart\_IT() in interrupt mode.

*Note:* HAL\_DMA2D\_BlendingStart() and HAL\_DMA2D\_BlendingStart\_IT() functions are used if the memory to memory with blending transfer mode is selected.

5. Optionally, configure and enable the CLUT using HAL\_DMA2D\_CLUTLoad() in polling mode or HAL\_DMA2D\_CLUTLoad\_IT() in interrupt mode.
6. Optionally, configure the line watermark in using the API HAL\_DMA2D\_ProgramLineEvent().
7. Optionally, configure the dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port in using the API HAL\_DMA2D\_ConfigDeadTime() and enable/disable the functionality with the APIs HAL\_DMA2D\_EnableDeadTime() or HAL\_DMA2D\_DisableDeadTime().
8. The transfer can be suspended, resumed and aborted using the following functions: HAL\_DMA2D\_Suspend(), HAL\_DMA2D\_Resume(), HAL\_DMA2D\_Abort().
9. The CLUT loading can be suspended, resumed and aborted using the following functions: HAL\_DMA2D\_CLUTLoading\_Suspend(), HAL\_DMA2D\_CLUTLoading\_Resume(), HAL\_DMA2D\_CLUTLoading\_Abort().
10. To control the DMA2D state, use the following function: HAL\_DMA2D\_GetState().
11. To read the DMA2D error code, use the following function: HAL\_DMA2D\_GetError().

#### **DMA2D HAL driver macros list**

Below the list of most used macros in DMA2D HAL driver :

- `__HAL_DMA2D_ENABLE`: Enable the DMA2D peripheral.
- `__HAL_DMA2D_GET_FLAG`: Get the DMA2D pending flags.
- `__HAL_DMA2D_CLEAR_FLAG`: Clear the DMA2D pending flags.
- `__HAL_DMA2D_ENABLE_IT`: Enable the specified DMA2D interrupts.
- `__HAL_DMA2D_DISABLE_IT`: Disable the specified DMA2D interrupts.
- `__HAL_DMA2D_GET_IT_SOURCE`: Check whether the specified DMA2D interrupt is enabled or not.

#### **Callback registration**

1. The compilation define `USE_HAL_DMA2D_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use function `@ref HAL_DMA2D_RegisterCallback()` to register a user callback.

2. Function @ref HAL\_DMA2D\_RegisterCallback() allows to register following callbacks: (+) XferCpltCallback : callback for transfer complete. (+) XferErrorCallback : callback for transfer error. (+) LineEventCallback : callback for line event. (+) CLUTLoadingCpltCallback : callback for CLUT loading completion. (+) MspInitCallback : DMA2D MspInit. (+) MspDeInitCallback : DMA2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
  3. Use function @ref HAL\_DMA2D\_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL\_DMA2D\_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) XferCpltCallback : callback for transfer complete. (+) XferErrorCallback : callback for transfer error. (+) LineEventCallback : callback for line event. (+) CLUTLoadingCpltCallback : callback for CLUT loading completion. (+) MspInitCallback : DMA2D MspInit. (+) MspDeInitCallback : DMA2D MspDeInit.
  4. By default, after the @ref HAL\_DMA2D\_Init and if the state is HAL\_DMA2D\_STATE\_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL\_DMA2D\_LineEventCallback(), @ref HAL\_DMA2D\_CLUTLoadingCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL\_DMA2D\_Init and @ref HAL\_DMA2D\_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL\_DMA2D\_Init and @ref HAL\_DMA2D\_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to. 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\_DMA2D\_RegisterCallback before calling @ref HAL\_DMA2D\_DeInit or @ref HAL\_DMA2D\_Init function. When The compilation define USE\_HAL\_DMA2D\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.
- (#) The compilation define USE\_HAL\_DMA2D\_REGISTER\_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL\_DMA2D\_RegisterCallback() to register a user callback. (#) Function @ref HAL\_DMA2D\_RegisterCallback() allows to register following callbacks:
- XferCpltCallback : callback for transfer complete.
  - XferErrorCallback : callback for transfer error.
  - LineEventCallback : callback for line event.
  - CLUTLoadingCpltCallback : callback for CLUT loading completion.
  - MspInitCallback : DMA2D MspInit.
  - MspDeInitCallback : DMA2D 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\_DMA2D\_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL\_DMA2D\_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
  - XferCpltCallback : callback for transfer complete.
  - XferErrorCallback : callback for transfer error.
  - LineEventCallback : callback for line event.
  - CLUTLoadingCpltCallback : callback for CLUT loading completion.
  - MspInitCallback : DMA2D MspInit.

- MspDelnitCallback : DMA2D MspDelnit. (#) By default, after the @ref HAL\_DMA2D\_Init and if the state is HAL\_DMA2D\_STATE\_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL\_DMA2D\_LineEventCallback(), @ref HAL\_DMA2D\_CLUTLoadingCpltCallback() Exception done for MspInit and MspDelnit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL\_DMA2D\_Init and @ref HAL\_DMA2D\_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDelnit are not null, the @ref HAL\_DMA2D\_Init and @ref HAL\_DMA2D\_DeInit keep and use the user MspInit/MspDelnit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDelnit 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/MspDelnit user callbacks using @ref HAL\_DMA2D\_RegisterCallback before calling @ref HAL\_DMA2D\_DeInit or @ref HAL\_DMA2D\_Init function. When The compilation define USE\_HAL\_DMA2D\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

*Note:* You can refer to the DMA2D HAL driver header file for more useful macros

## 20.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D

This section contains the following APIs:

- [HAL\\_DMA2D\\_Init\(\)](#)
- [HAL\\_DMA2D\\_DeInit\(\)](#)
- [HAL\\_DMA2D\\_MspInit\(\)](#)
- [HAL\\_DMA2D\\_MspDeInit\(\)](#)

## 20.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size then start the DMA2D transfer.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size then start the DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer with interrupt.
- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Resume DMA2D transfer.
- Enable CLUT transfer.
- Configure CLUT loading then start transfer in polling mode.
- Configure CLUT loading then start transfer in interrupt mode.
- Abort DMA2D CLUT loading.
- Suspend DMA2D CLUT loading.
- Resume DMA2D CLUT loading.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- Transfer watermark callback.
- CLUT Transfer Complete callback.

This section contains the following APIs:

- [HAL\\_DMA2D\\_Start\(\)](#)
- [HAL\\_DMA2D\\_Start\\_IT\(\)](#)

- *HAL\_DMA2D\_BlendingStart()*
- *HAL\_DMA2D\_BlendingStart\_IT()*
- *HAL\_DMA2D\_Abort()*
- *HAL\_DMA2D\_Suspend()*
- *HAL\_DMA2D\_Resume()*
- *HAL\_DMA2D\_EnableCLUT()*
- *HAL\_DMA2D\_CLUTStartLoad()*
- *HAL\_DMA2D\_CLUTStartLoad\_IT()*
- *HAL\_DMA2D\_CLUTLoad()*
- *HAL\_DMA2D\_CLUTLoad\_IT()*
- *HAL\_DMA2D\_CLUTLoading\_Abort()*
- *HAL\_DMA2D\_CLUTLoading\_Suspend()*
- *HAL\_DMA2D\_CLUTLoading\_Resume()*
- *HAL\_DMA2D\_PollForTransfer()*
- *HAL\_DMA2D\_IRQHandler()*
- *HAL\_DMA2D\_LineEventCallback()*
- *HAL\_DMA2D\_CLUTLoadingCpltCallback()*

#### 20.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or background layer parameters.
- Configure the DMA2D CLUT transfer.
- Configure the line watermark
- Configure the dead time value.
- Enable or disable the dead time value functionality.

This section contains the following APIs:

- *HAL\_DMA2D\_ConfigLayer()*
- *HAL\_DMA2D\_ConfigCLUT()*
- *HAL\_DMA2D\_ProgramLineEvent()*
- *HAL\_DMA2D\_EnableDeadTime()*
- *HAL\_DMA2D\_DisableDeadTime()*
- *HAL\_DMA2D\_ConfigDeadTime()*

#### 20.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to:

- Get the DMA2D state
- Get the DMA2D error code

This section contains the following APIs:

- *HAL\_DMA2D\_GetState()*
- *HAL\_DMA2D\_GetError()*

#### 20.2.6 Detailed description of functions

##### HAL\_DMA2D\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_Init (DMA2D\_HandleTypeDef \* hdma2d)**

###### Function description

Initialize the DMA2D according to the specified parameters in the DMA2D\_InitTypeDef and create the associated handle.



**Parameters**

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

**Return values**

- **HAL**: status

**HAL\_DMA2D\_DeInit**
**Function name**
**HAL\_StatusTypeDef HAL\_DMA2D\_DeInit (DMA2D\_HandleTypeDef \* hdma2d)**
**Function description**

Deinitializes the DMA2D peripheral registers to their default reset values.

**Parameters**

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

**Return values**

- **None**:

**HAL\_DMA2D\_MspInit**
**Function name**
**void HAL\_DMA2D\_MspInit (DMA2D\_HandleTypeDef \* hdma2d)**
**Function description**

Initializes the DMA2D MSP.

**Parameters**

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

**Return values**

- **None**:

**HAL\_DMA2D\_MspDeInit**
**Function name**
**void HAL\_DMA2D\_MspDeInit (DMA2D\_HandleTypeDef \* hdma2d)**
**Function description**

Deinitializes the DMA2D MSP.

**Parameters**

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

**Return values**

- **None**:

**HAL\_DMA2D\_Start**
**Function name**
**HAL\_StatusTypeDef HAL\_DMA2D\_Start (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t pdata, uint32\_t DstAddress, uint32\_t Width, uint32\_t Height)**

### Function description

Start the DMA2D Transfer.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata**: Configure the source memory Buffer address if Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress**: The destination memory Buffer address.
- **Width**: The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height**: The height of data to be transferred from source to destination (expressed in number of lines).

### Return values

- **HAL**: status

### HAL\_DMA2D\_BlendingStart

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_BlendingStart (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t SrcAddress1, uint32\_t SrcAddress2, uint32\_t DstAddress, uint32\_t Width, uint32\_t Height)**

### Function description

Start the multi-source DMA2D Transfer.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1**: The source memory Buffer address for the foreground layer.
- **SrcAddress2**: The source memory Buffer address for the background layer.
- **DstAddress**: The destination memory Buffer address.
- **Width**: The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height**: The height of data to be transferred from source to destination (expressed in number of lines).

### Return values

- **HAL**: status

### HAL\_DMA2D\_Start\_IT

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_Start\_IT (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t pdata, uint32\_t DstAddress, uint32\_t Width, uint32\_t Height)**

### Function description

Start the DMA2D Transfer with interrupt enabled.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata**: Configure the source memory Buffer address if the Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress**: The destination memory Buffer address.
- **Width**: The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height**: The height of data to be transferred from source to destination (expressed in number of lines).

### Return values

- **HAL**: status

#### HAL\_DMA2D\_BlendingStart\_IT

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_BlendingStart\_IT (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t SrcAddress1, uint32\_t SrcAddress2, uint32\_t DstAddress, uint32\_t Width, uint32\_t Height)**

### Function description

Start the multi-source DMA2D Transfer with interrupt enabled.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1**: The source memory Buffer address for the foreground layer.
- **SrcAddress2**: The source memory Buffer address for the background layer.
- **DstAddress**: The destination memory Buffer address.
- **Width**: The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height**: The height of data to be transferred from source to destination (expressed in number of lines).

### Return values

- **HAL**: status

#### HAL\_DMA2D\_Suspend

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_Suspend (DMA2D\_HandleTypeDef \* hdma2d)**

### Function description

Suspend the DMA2D Transfer.

### Parameters

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

### Return values

- **HAL**: status

#### HAL\_DMA2D\_Resume

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_Resume (DMA2D\_HandleTypeDef \* hdma2d)**

### Function description

Resume the DMA2D Transfer.

### Parameters

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

### Return values

- **HAL**: status

### HAL\_DMA2D\_Abort

### Function name

HAL\_StatusTypeDef HAL\_DMA2D\_Abort (DMA2D\_HandleTypeDef \* hdma2d)

### Function description

Abort the DMA2D Transfer.

### Parameters

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

### Return values

- **HAL**: status

### HAL\_DMA2D\_EnableCLUT

### Function name

HAL\_StatusTypeDef HAL\_DMA2D\_EnableCLUT (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t LayerIdx)

### Function description

Enable the DMA2D CLUT Transfer.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx**: DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL**: status

### HAL\_DMA2D\_CLUTStartLoad

### Function name

HAL\_StatusTypeDef HAL\_DMA2D\_CLUTStartLoad (DMA2D\_HandleTypeDef \* hdma2d, DMA2D\_CLUTCfgTypeDef \* CLUTCfg, uint32\_t LayerIdx)

### Function description

Start DMA2D CLUT Loading.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg**: Pointer to a DMA2D\_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx**: DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL:** status

**HAL\_DMA2D\_CLUTStartLoad\_IT**

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_CLUTStartLoad\_IT (DMA2D\_HandleTypeDef \* hdma2d, DMA2D\_CLUTCfgTypeDef \* CLUTCfg, uint32\_t LayerIdx)**

### Function description

Start DMA2D CLUT Loading with interrupt enabled.

### Parameters

- **hdma2d:** Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D\_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL:** status

**HAL\_DMA2D\_CLUTLoad**

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_CLUTLoad (DMA2D\_HandleTypeDef \* hdma2d, DMA2D\_CLUTCfgTypeDef CLUTCfg, uint32\_t LayerIdx)**

### Function description

Start DMA2D CLUT Loading.

### Parameters

- **hdma2d:** Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D\_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL:** status

### Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL\_DMA2D\_CLUTStartLoad() instead to benefit from code compactness, code size and improved heap usage.

**HAL\_DMA2D\_CLUTLoad\_IT**

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_CLUTLoad\_IT (DMA2D\_HandleTypeDef \* hdma2d, DMA2D\_CLUTCfgTypeDef CLUTCfg, uint32\_t LayerIdx)**

### Function description

Start DMA2D CLUT Loading with interrupt enabled.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg**: Pointer to a DMA2D\_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx**: DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL**: status

### Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL\_DMA2D\_CLUTStartLoad\_IT() instead to benefit from code compactness, code size and improved heap usage.

#### HAL\_DMA2D\_CLUTLoading\_Abort

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_CLUTLoading\_Abort (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t LayerIdx)**

### Function description

Abort the DMA2D CLUT loading.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx**: DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL**: status

#### HAL\_DMA2D\_CLUTLoading\_Suspend

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_CLUTLoading\_Suspend (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t LayerIdx)**

### Function description

Suspend the DMA2D CLUT loading.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx**: DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL**: status

#### HAL\_DMA2D\_CLUTLoading\_Resume

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_CLUTLoading\_Resume (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t LayerIdx)**

### Function description

Resume the DMA2D CLUT loading.

### Parameters

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx**: DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

### Return values

- **HAL**: status

### HAL\_DMA2D\_PollForTransfer

### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_PollForTransfer (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t Timeout)**

### Function description

Polling for transfer complete or CLUT loading.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

### HAL\_DMA2D\_IRQHandler

### Function name

**void HAL\_DMA2D\_IRQHandler (DMA2D\_HandleTypeDef \* hdma2d)**

### Function description

Handle DMA2D interrupt request.

### Parameters

- **hdma2d**: Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

### Return values

- **HAL**: status

### HAL\_DMA2D\_LineEventCallback

### Function name

**void HAL\_DMA2D\_LineEventCallback (DMA2D\_HandleTypeDef \* hdma2d)**

### Function description

Transfer watermark callback.

### Parameters

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

### Return values

- **None**:

### HAL\_DMA2D\_CLUTLoadingCpltCallback

#### Function name

**void HAL\_DMA2D\_CLUTLoadingCpltCallback (DMA2D\_HandleTypeDef \* hdma2d)**

#### Function description

CLUT Transfer Complete callback.

#### Parameters

- **hdma2d:** pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

#### Return values

- **None:**

### HAL\_DMA2D\_ConfigLayer

#### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_ConfigLayer (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t LayerIdx)**

#### Function description

Configure the DMA2D Layer according to the specified parameters in the DMA2D\_HandleTypeDef.

#### Parameters

- **hdma2d:** Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

#### Return values

- **HAL:** status

### HAL\_DMA2D\_ConfigCLUT

#### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_ConfigCLUT (DMA2D\_HandleTypeDef \* hdma2d, DMA2D\_CLUTCfgTypeDef CLUTCfg, uint32\_t LayerIdx)**

#### Function description

Configure the DMA2D CLUT Transfer.

#### Parameters

- **hdma2d:** Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D\_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D\_BACKGROUND\_LAYER(0) / DMA2D\_FOREGROUND\_LAYER(1)

#### Return values

- **HAL:** status

#### Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL\_DMA2D\_CLUTStartLoad() instead to benefit from code compactness, code size and improved heap usage.



### HAL\_DMA2D\_ProgramLineEvent

#### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_ProgramLineEvent (DMA2D\_HandleTypeDef \* hdma2d, uint32\_t Line)**

#### Function description

Configure the line watermark.

#### Parameters

- **hdma2d:** Pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Line:** Line Watermark configuration (maximum 16-bit long value expected).

#### Return values

- **HAL:** status

#### Notes

- HAL\_DMA2D\_ProgramLineEvent() API enables the transfer watermark interrupt.
- The transfer watermark interrupt is disabled once it has occurred.

### HAL\_DMA2D\_EnableDeadTime

#### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_EnableDeadTime (DMA2D\_HandleTypeDef \* hdma2d)**

#### Function description

Enable DMA2D dead time feature.

#### Parameters

- **hdma2d:** DMA2D handle.

#### Return values

- **HAL:** status

### HAL\_DMA2D\_DisableDeadTime

#### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_DisableDeadTime (DMA2D\_HandleTypeDef \* hdma2d)**

#### Function description

Disable DMA2D dead time feature.

#### Parameters

- **hdma2d:** DMA2D handle.

#### Return values

- **HAL:** status

### HAL\_DMA2D\_ConfigDeadTime

#### Function name

**HAL\_StatusTypeDef HAL\_DMA2D\_ConfigDeadTime (DMA2D\_HandleTypeDef \* hdma2d, uint8\_t DeadTime)**

#### Function description

Configure dead time.

**Parameters**

- **hdma2d**: DMA2D handle.
- **DeadTime**: dead time value.

**Return values**

- **HAL**: status

**Notes**

- The dead time value represents the guaranteed minimum number of cycles between two consecutive transactions on the AHB bus.

**HAL\_DMA2D\_GetState**
**Function name**

**HAL\_DMA2D\_StateTypeDef HAL\_DMA2D\_GetState (DMA2D\_HandleTypeDef \* hdma2d)**

**Function description**

Return the DMA2D state.

**Parameters**

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

**Return values**

- **HAL**: state

**HAL\_DMA2D\_GetError**
**Function name**

**uint32\_t HAL\_DMA2D\_GetError (DMA2D\_HandleTypeDef \* hdma2d)**

**Function description**

Return the DMA2D error code.

**Parameters**

- **hdma2d**: pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for DMA2D.

**Return values**

- **DMA2D**: Error Code

## 20.3 DMA2D Firmware driver defines

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

### 20.3.1 DMA2D

DMA2D

#### *DMA2D API Aliases*

#### HAL\_DMA2D\_DisableCLUT

Aliased to HAL\_DMA2D\_CLUTLoading\_Abort for compatibility with legacy code

#### *DMA2D Alpha Mode*

#### DMA2D\_NO\_MODIF\_ALPHA

No modification of the alpha channel value

#### DMA2D\_REPLACE\_ALPHA

Replace original alpha channel value by programmed alpha value

**DMA2D\_COMBINE\_ALPHA**

Replace original alpha channel value by programmed alpha value with original alpha channel value

***DMA2D CLUT Color Mode***

**DMA2D\_CCM\_ARGB8888**

ARGB8888 DMA2D CLUT color mode

**DMA2D\_CCM\_RGB888**

RGB888 DMA2D CLUT color mode

***DMA2D CLUT Size***

**DMA2D\_CLUT\_SIZE**

DMA2D maximum CLUT size

***DMA2D Color Value***

**DMA2D\_COLOR\_VALUE**

Color value mask

***DMA2D Error Code***

**HAL\_DMA2D\_ERROR\_NONE**

No error

**HAL\_DMA2D\_ERROR\_TE**

Transfer error

**HAL\_DMA2D\_ERROR\_CE**

Configuration error

**HAL\_DMA2D\_ERROR\_CAE**

CLUT access error

**HAL\_DMA2D\_ERROR\_TIMEOUT**

Timeout error

***DMA2D Exported Macros***

**\_\_HAL\_DMA2D\_RESET\_HANDLE\_STATE**
**Description:**

- Reset DMA2D handle state.

**Parameters:**

- `__HANDLE__`: specifies the DMA2D handle.

**Return value:**

- None

**\_\_HAL\_DMA2D\_ENABLE**
**Description:**

- Enable the DMA2D.

**Parameters:**

- `__HANDLE__`: DMA2D handle

**Return value:**

- None.

### **\_\_HAL\_DMA2D\_GET\_FLAG**

**Description:**

- Get the DMA2D pending flags.

**Parameters:**

- `__HANDLE__`: DMA2D handle
- `__FLAG__`: flag to check. This parameter can be any combination of the following values:
  - `DMA2D_FLAG_CE`: Configuration error flag
  - `DMA2D_FLAG CTC`: CLUT transfer complete flag
  - `DMA2D_FLAG CAE`: CLUT access error flag
  - `DMA2D_FLAG TW`: Transfer Watermark flag
  - `DMA2D_FLAG TC`: Transfer complete flag
  - `DMA2D_FLAG TE`: Transfer error flag

**Return value:**

- The: state of FLAG.

### **\_\_HAL\_DMA2D\_CLEAR\_FLAG**

**Description:**

- Clear the DMA2D pending flags.

**Parameters:**

- `__HANDLE__`: DMA2D handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DMA2D_FLAG_CE`: Configuration error flag
  - `DMA2D_FLAG CTC`: CLUT transfer complete flag
  - `DMA2D_FLAG CAE`: CLUT access error flag
  - `DMA2D_FLAG TW`: Transfer Watermark flag
  - `DMA2D_FLAG TC`: Transfer complete flag
  - `DMA2D_FLAG TE`: Transfer error flag

**Return value:**

- None

### **\_\_HAL\_DMA2D\_ENABLE\_IT**

**Description:**

- Enable the specified DMA2D interrupts.

**Parameters:**

- `__HANDLE__`: DMA2D handle
- `__INTERRUPT__`: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
  - `DMA2D_IT_CE`: Configuration error interrupt mask
  - `DMA2D_IT CTC`: CLUT transfer complete interrupt mask
  - `DMA2D_IT CAE`: CLUT access error interrupt mask
  - `DMA2D_IT TW`: Transfer Watermark interrupt mask
  - `DMA2D_IT TC`: Transfer complete interrupt mask
  - `DMA2D_IT TE`: Transfer error interrupt mask

**Return value:**

- None

### **\_\_HAL\_DMA2D\_DISABLE\_IT**

**Description:**

- Disable the specified DMA2D interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: DMA2D handle
- **\_\_INTERRUPT\_\_**: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
  - **DMA2D\_IT\_CE**: Configuration error interrupt mask
  - **DMA2D\_IT\_CTC**: CLUT transfer complete interrupt mask
  - **DMA2D\_IT\_CAE**: CLUT access error interrupt mask
  - **DMA2D\_IT\_TW**: Transfer Watermark interrupt mask
  - **DMA2D\_IT\_TC**: Transfer complete interrupt mask
  - **DMA2D\_IT\_TE**: Transfer error interrupt mask

**Return value:**

- None

### **\_\_HAL\_DMA2D\_GET\_IT\_SOURCE**

**Description:**

- Check whether the specified DMA2D interrupt source is enabled or not.

**Parameters:**

- **\_\_HANDLE\_\_**: DMA2D handle
- **\_\_INTERRUPT\_\_**: specifies the DMA2D interrupt source to check. This parameter can be one of the following values:
  - **DMA2D\_IT\_CE**: Configuration error interrupt mask
  - **DMA2D\_IT\_CTC**: CLUT transfer complete interrupt mask
  - **DMA2D\_IT\_CAE**: CLUT access error interrupt mask
  - **DMA2D\_IT\_TW**: Transfer Watermark interrupt mask
  - **DMA2D\_IT\_TC**: Transfer complete interrupt mask
  - **DMA2D\_IT\_TE**: Transfer error interrupt mask

**Return value:**

- The: state of INTERRUPT source.

***DMA2D Exported Types***

### **MAX\_DMA2D\_LAYER**

DMA2D maximum number of layers

***DMA2D Flags***

### **DMA2D\_FLAG\_CE**

Configuration Error Interrupt Flag

### **DMA2D\_FLAG\_CTC**

CLUT Transfer Complete Interrupt Flag

### **DMA2D\_FLAG\_CAE**

CLUT Access Error Interrupt Flag

### **DMA2D\_FLAG\_TW**

Transfer Watermark Interrupt Flag

### **DMA2D\_FLAG\_TC**

Transfer Complete Interrupt Flag

**DMA2D\_FLAG\_TE**

Transfer Error Interrupt Flag  
**DMA2D Input Color Mode**

**DMA2D\_INPUT\_ARGB8888**

ARGB8888 color mode

**DMA2D\_INPUT\_RGB888**

RGB888 color mode

**DMA2D\_INPUT\_RGB565**

RGB565 color mode

**DMA2D\_INPUT\_ARGB1555**

ARGB1555 color mode

**DMA2D\_INPUT\_ARGB4444**

ARGB4444 color mode

**DMA2D\_INPUT\_L8**

L8 color mode

**DMA2D\_INPUT\_AL44**

AL44 color mode

**DMA2D\_INPUT\_AL88**

AL88 color mode

**DMA2D\_INPUT\_L4**

L4 color mode

**DMA2D\_INPUT\_A8**

A8 color mode

**DMA2D\_INPUT\_A4**

A4 color mode

**DMA2D Interrupts**

**DMA2D\_IT\_CE**

Configuration Error Interrupt

**DMA2D\_IT\_CTC**

CLUT Transfer Complete Interrupt

**DMA2D\_IT\_CAE**

CLUT Access Error Interrupt

**DMA2D\_IT\_TW**

Transfer Watermark Interrupt

**DMA2D\_IT\_TC**

Transfer Complete Interrupt

**DMA2D\_IT\_TE**

Transfer Error Interrupt

**DMA2D Layers**

#### DMA2D\_BACKGROUND\_LAYER

DMA2D Background Layer (layer 0)

#### DMA2D\_FOREGROUND\_LAYER

DMA2D Foreground Layer (layer 1)

**DMA2D Maximum Line Watermark**

#### DMA2D\_LINE\_WATERMARK\_MAX

DMA2D maximum line watermark

**DMA2D Maximum Number of Layers**

#### DMA2D\_MAX\_LAYER

DMA2D maximum number of layers

**DMA2D Mode**

#### DMA2D\_M2M

DMA2D memory to memory transfer mode

#### DMA2D\_M2M\_PFC

DMA2D memory to memory with pixel format conversion transfer mode

#### DMA2D\_M2M\_BLEND

DMA2D memory to memory with blending transfer mode

#### DMA2D\_R2M

DMA2D register to memory transfer mode

**DMA2D Offset**

#### DMA2D\_OFFSET

maximum Line Offset

**DMA2D Output Color Mode**

#### DMA2D\_OUTPUT\_ARGB8888

ARGB8888 DMA2D color mode

#### DMA2D\_OUTPUT\_RGB888

RGB888 DMA2D color mode

#### DMA2D\_OUTPUT\_RGB565

RGB565 DMA2D color mode

#### DMA2D\_OUTPUT\_ARGB1555

ARGB1555 DMA2D color mode

#### DMA2D\_OUTPUT\_ARGB4444

ARGB4444 DMA2D color mode

**DMA2D Size**

#### DMA2D\_PIXEL

DMA2D maximum number of pixels per line

#### DMA2D\_LINE

DMA2D maximum number of lines

**DMA2D Time Out**

#### DMA2D\_TIMEOUT\_ABORT

1s

DMA2D\_TIMEOUT\_SUSPEND

1s



## 21 HAL DMA Generic Driver

### 21.1 DMA Firmware driver registers structures

#### 21.1.1 DMA\_InitTypeDef

*DMA\_InitTypeDef* is defined in the `stm32f4xx_hal_dma.h`

##### Data Fields

- *uint32\_t Channel*
- *uint32\_t Direction*
- *uint32\_t PeriphInc*
- *uint32\_t MemInc*
- *uint32\_t PeriphDataAlignment*
- *uint32\_t MemDataAlignment*
- *uint32\_t Mode*
- *uint32\_t Priority*
- *uint32\_t FIFOMode*
- *uint32\_t FIFOThreshold*
- *uint32\_t MemBurst*
- *uint32\_t PeriphBurst*

##### Field Documentation

- *uint32\_t DMA\_InitTypeDef::Channel*  
Specifies the channel used for the specified stream. This parameter can be a value of [DMA\\_Channel\\_selection](#)
- *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 Streamx. 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 Stream
- *uint32\_t DMA\_InitTypeDef::Priority*  
Specifies the software priority for the DMAy Streamx. This parameter can be a value of [DMA\\_Priority\\_level](#)
- *uint32\_t DMA\_InitTypeDef::FIFOMode*  
Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of [DMA\\_FIFO\\_direct\\_mode](#)

##### Note:

- The Direct mode (FIFO mode disabled) cannot be used if the memory-to-memory data transfer is configured on the selected stream

- **`uint32_t DMA_InitTypeDef::FIFOThreshold`**  
Specifies the FIFO threshold level. This parameter can be a value of [DMA\\_FIFO\\_threshold\\_level](#)
- **`uint32_t DMA_InitTypeDef::MemBurst`**  
Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of [DMA\\_Memory\\_burst](#)  
**Note:**
  - The burst mode is possible only if the address Increment mode is enabled.
- **`uint32_t DMA_InitTypeDef::PeriphBurst`**  
Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of [DMA\\_Peripheral\\_burst](#)  
**Note:**
  - The burst mode is possible only if the address Increment mode is enabled.

### 21.1.2 `__DMA_HandleTypeDef`

`__DMA_HandleTypeDef` is defined in the `stm32f4xx_hal_dma.h`

#### Data Fields

- **`DMA_Stream_TypeDef * Instance`**
- **`DMA_InitTypeDef Init`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_DMA_StateTypeDef State`**
- **`void * Parent`**
- **`void(* XferCpltCallback`**
- **`void(* XferHalfCpltCallback`**
- **`void(* XferM1CpltCallback`**
- **`void(* XferM1HalfCpltCallback`**
- **`void(* XferErrorCallback`**
- **`void(* XferAbortCallback`**
- **`__IO uint32_t ErrorCode`**
- **`uint32_t StreamBaseAddress`**
- **`uint32_t StreamIndex`**

#### Field Documentation

- **`DMA_Stream_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::XferM1CpltCallback)(struct __DMA_HandleTypeDef *hdma)`**  
DMA transfer complete Memory1 callback
- **`void(* __DMA_HandleTypeDef::XferM1HalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)`**  
DMA transfer Half complete Memory1 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
- **`uint32_t __DMA_HandleTypeDef::StreamBaseAddress`**  
DMA Stream Base Address
- **`uint32_t __DMA_HandleTypeDef::StreamIndex`**  
DMA Stream Index

## 21.2 DMA Firmware driver API description

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

### 21.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Stream (except for internal SRAM/FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests.
2. For a given Stream, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular, Normal or peripheral flow control mode, Stream Priority level, Source and Destination Increment mode, FIFO mode and its Threshold (if needed), Burst mode for Source and/or Destination (if needed) using `HAL_DMA_Init()` function.

*Note:* Prior to `HAL_DMA_Init()` the clock must be enabled for DMA through the following macros: `__HAL_RCC_DMA1_CLK_ENABLE()` or `__HAL_RCC_DMA2_CLK_ENABLE()`.

#### 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.
- Use `HAL_DMA_Abort()` function to abort the current transfer.

#### 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 by customization of function pointer `XferCpltCallback` and `XferErrorCallback` (i.e a member of DMA handle structure).
1. Use `HAL_DMA_GetState()` function to return the DMA state and `HAL_DMA_GetError()` in case of error detection.
  2. Use `HAL_DMA_Abort_IT()` function to abort the current transfer

*Note:* In Memory-to-Memory transfer mode, Circular mode is not allowed.

*Note:* The FIFO is used mainly to reduce bus usage and to allow data packing/unpacking: it is possible to set different Data Sizes for the Peripheral and the Memory (ie. you can set Half-Word data size for the peripheral to access its data register and set Word data size for the Memory to gain in access time. Each two half words will be packed and written in a single access to a Word in the Memory).

*Note:* When FIFO is disabled, it is not allowed to configure different Data Sizes for Source and Destination. In this case the Peripheral Data Size will be applied to both Source and Destination.

### DMA HAL driver macros list

Below the list of most used macros in DMA HAL driver.

- `__HAL_DMA_ENABLE`: Enable the specified DMA Stream.
- `__HAL_DMA_DISABLE`: Disable the specified DMA Stream.
- `__HAL_DMA_GET_IT_SOURCE`: Check whether the specified DMA Stream interrupt has occurred or not.

*Note:* You can refer to the DMA HAL driver header file for more useful macros

### 21.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Stream source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Stream 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\(\)\*](#)

### 21.2.3 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\(\)\*](#)
- [\*HAL\\_DMA\\_CleanCallbacks\(\)\*](#)

### 21.2.4 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\(\)\*](#)

### 21.2.5 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 create the associated handle.

### Parameters

- **hdma:** Pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

### Return values

- **HAL:** status

**HAL\_DMA\_DeInit**

### Function name

**HAL\_StatusTypeDef HAL\_DMA\_DeInit (DMA\_HandleTypeDef \* hdma)**

### Function description

DeInitializes the DMA peripheral.

### Parameters

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

### 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

Starts the DMA Transfer.

### Parameters

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

### 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 Stream.
- **SrcAddress**: The source memory Buffer address
- **DstAddress**: The destination memory Buffer address
- **DataLength**: The length of data to be transferred from source to destination

### Return values

- **HAL**: status

### HAL\_DMA\_Abort

### Function name

**HAL\_StatusTypeDef HAL\_DMA\_Abort (DMA\_HandleTypeDef \* hdma)**

### Function description

Aborts the DMA Transfer.

### Parameters

- **hdma**: pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

### Return values

- **HAL**: status

### Notes

- After disabling a DMA Stream, a check for wait until the DMA Stream is effectively disabled is added. If a Stream is disabled while a data transfer is ongoing, the current data will be transferred and the Stream will be effectively disabled only after the transfer of this single data is finished.

### 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 Stream.

### 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 Stream.
- **CompleteLevel**: Specifies the DMA level complete.
- **Timeout**: Timeout duration.

### Return values

- **HAL**: status

### Notes

- The polling mode is kept in this version for legacy. it is recommended to use the IT model instead. This model could be used for debug purpose.
- The HAL\_DMA\_PollForTransfer API cannot be used in circular and double buffering mode (automatic circular mode).

### HAL\_DMA\_IRQHandler

#### Function name

**void HAL\_DMA\_IRQHandler (DMA\_HandleTypeDef \* hdma)**

#### Function description

Handles DMA interrupt request.

#### Parameters

- **hdma**: pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

#### Return values

- **None**:

### HAL\_DMA\_CleanCallbacks

#### Function name

**HAL\_StatusTypeDef HAL\_DMA\_CleanCallbacks (DMA\_HandleTypeDef \* hdma)**

#### Function description

### 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 Stream.
- **CallbackID**: User Callback identifier a DMA\_HandleTypeDef structure as parameter.
- **pCallback**: pointer to private callback 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 Stream.
- **CallbackID**: User Callback identifier 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

Returns the DMA state.

#### Parameters

- **hdma**: pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

#### 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 Stream.

#### Return values

- **DMA**: Error Code

## 21.3 DMA Firmware driver defines

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

### 21.3.1 DMA

DMA

#### *DMA Channel selection*

#### DMA\_CHANNEL\_0

DMA Channel 0



**DMA\_CHANNEL\_1**

DMA Channel 1

**DMA\_CHANNEL\_2**

DMA Channel 2

**DMA\_CHANNEL\_3**

DMA Channel 3

**DMA\_CHANNEL\_4**

DMA Channel 4

**DMA\_CHANNEL\_5**

DMA Channel 5

**DMA\_CHANNEL\_6**

DMA Channel 6

**DMA\_CHANNEL\_7**

DMA Channel 7

***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\_FE**

FIFO error

**HAL\_DMA\_ERROR\_DME**

Direct Mode error

**HAL\_DMA\_ERROR\_TIMEOUT**

Timeout error

**HAL\_DMA\_ERROR\_PARAM**

Parameter error

**HAL\_DMA\_ERROR\_NO\_XFER**

Abort requested with no Xfer ongoing

**HAL\_DMA\_ERROR\_NOT\_SUPPORTED**

Not supported mode

***DMA FIFO direct mode***

**DMA\_FIFOMODE\_DISABLE**

FIFO mode disable

**DMA\_FIFOMODE\_ENABLE**

FIFO mode enable

***DMA FIFO threshold level*****DMA\_FIFO\_THRESHOLD\_1QUARTERFULL**

FIFO threshold 1 quart full configuration

**DMA\_FIFO\_THRESHOLD\_HALFFULL**

FIFO threshold half full configuration

**DMA\_FIFO\_THRESHOLD\_3QUARTERSFULL**

FIFO threshold 3 quarts full configuration

**DMA\_FIFO\_THRESHOLD\_FULL**

FIFO threshold full configuration

***DMA flag definitions*****DMA\_FLAG\_FEIF0\_4****DMA\_FLAG\_DMEIF0\_4****DMA\_FLAG\_TEIF0\_4****DMA\_FLAG\_HTIF0\_4****DMA\_FLAG\_TCIF0\_4****DMA\_FLAG\_FEIF1\_5****DMA\_FLAG\_DMEIF1\_5****DMA\_FLAG\_TEIF1\_5****DMA\_FLAG\_HTIF1\_5****DMA\_FLAG\_TCIF1\_5****DMA\_FLAG\_FEIF2\_6****DMA\_FLAG\_DMEIF2\_6****DMA\_FLAG\_TEIF2\_6****DMA\_FLAG\_HTIF2\_6****DMA\_FLAG\_TCIF2\_6****DMA\_FLAG\_FEIF3\_7****DMA\_FLAG\_DMEIF3\_7****DMA\_FLAG\_TEIF3\_7**

DMA\_FLAG\_HTIF3\_7

DMA\_FLAG\_TCIF3\_7

***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\_IT\_DME

DMA\_IT\_FE

***DMA Memory burst***

DMA\_MBURST\_SINGLE

DMA\_MBURST\_INC4

DMA\_MBURST\_INC8

DMA\_MBURST\_INC16

***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\_PFCTRL**

Peripheral flow control mode

***DMA Peripheral burst*****DMA\_PBURST\_SINGLE****DMA\_PBURST\_INC4****DMA\_PBURST\_INC8****DMA\_PBURST\_INC16*****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

## 22 HAL DMA Extension Driver

### 22.1 DMAEx Firmware driver API description

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

#### 22.1.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

1. Start a multi buffer transfer using the `HAL_DMA_MultiBufferStart()` function for polling mode or `HAL_DMA_MultiBufferStart_IT()` for interrupt mode.

*Note:* In Memory-to-Memory transfer mode, Multi (Double) Buffer mode is not allowed.

*Note:* When Multi (Double) Buffer mode is enabled the, transfer is circular by default.

*Note:* In Multi (Double) buffer mode, it is possible to update the base address for the AHB memory port on the fly (`DMA_SxM0AR` or `DMA_SxM1AR`) when the stream is enabled.

#### 22.1.2 Extended features functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start MultiBuffer DMA transfer
- Configure the source, destination address and data length and Start MultiBuffer DMA transfer with interrupt
- Change on the fly the memory0 or memory1 address.

This section contains the following APIs:

- [HAL\\_DMAEx\\_MultiBufferStart\(\)](#)
- [HAL\\_DMAEx\\_MultiBufferStart\\_IT\(\)](#)
- [HAL\\_DMAEx\\_ChangeMemory\(\)](#)

#### 22.1.3 Detailed description of functions

##### HAL\_DMAEx\_MultiBufferStart

###### Function name

`HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)`

###### Function description

Starts the multi\_buffer DMA Transfer.

###### Parameters

- **hdma:** pointer to a `DMA_HandleTypeDef` structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

###### Return values

- **HAL:** status

##### HAL\_DMAEx\_MultiBufferStart\_IT

###### Function name

`HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)`

### Function description

Starts the multi\_buffer DMA Transfer with interrupt enabled.

### Parameters

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

### Return values

- **HAL:** status

### HAL\_DMAEx\_ChangeMemory

### Function name

```
HAL_StatusTypeDef HAL_DMAEx_ChangeMemory (DMA_HandleTypeDef * hdma, uint32_t Address,
HAL_DMA_MemoryTypeDef memory)
```

### Function description

Change the memory0 or memory1 address on the fly.

### Parameters

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **Address:** The new address
- **memory:** the memory to be changed, This parameter can be one of the following values: MEMORY0 / MEMORY1

### Return values

- **HAL:** status

### Notes

- The MEMORY0 address can be changed only when the current transfer use MEMORY1 and the MEMORY1 address can be changed only when the current transfer use MEMORY0.

## 23 HAL DSI Generic Driver

### 23.1 DSI Firmware driver registers structures

#### 23.1.1 DSI\_InitTypeDef

*DSI\_InitTypeDef* is defined in the stm32f4xx\_hal\_dsi.h

Data Fields

- *uint32\_t AutomaticClockLaneControl*
- *uint32\_t TXEscapeCkdiv*
- *uint32\_t NumberOfLanes*

Field Documentation

- *uint32\_t DSI\_InitTypeDef::AutomaticClockLaneControl*  
Automatic clock lane control This parameter can be any value of *DSI\_Automatic\_Clk\_Lane\_Control*
- *uint32\_t DSI\_InitTypeDef::TXEscapeCkdiv*  
TX Escape clock division The values 0 and 1 stop the TX\_ESC clock generation
- *uint32\_t DSI\_InitTypeDef::NumberOfLanes*  
Number of lanes This parameter can be any value of *DSI\_Number\_Of\_Lanes*

#### 23.1.2 DSI\_PLLInitTypeDef

*DSI\_PLLInitTypeDef* is defined in the stm32f4xx\_hal\_dsi.h

Data Fields

- *uint32\_t PLLNDIV*
- *uint32\_t PLLIDF*
- *uint32\_t PLLODF*

Field Documentation

- *uint32\_t DSI\_PLLInitTypeDef::PLLNDIV*  
PLL Loop Division Factor This parameter must be a value between 10 and 125
- *uint32\_t DSI\_PLLInitTypeDef::PLLIDF*  
PLL Input Division Factor This parameter can be any value of *DSI\_PLL\_IDF*
- *uint32\_t DSI\_PLLInitTypeDef::PLLODF*  
PLL Output Division Factor This parameter can be any value of *DSI\_PLL\_ODF*

#### 23.1.3 DSI\_VidCfgTypeDef

*DSI\_VidCfgTypeDef* is defined in the stm32f4xx\_hal\_dsi.h

Data Fields

- *uint32\_t VirtualChannelID*
- *uint32\_t ColorCoding*
- *uint32\_t LooselyPacked*
- *uint32\_t Mode*
- *uint32\_t PacketSize*
- *uint32\_t NumberOfChunks*
- *uint32\_t NullPacketSize*
- *uint32\_t HSPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t DEPolarity*
- *uint32\_t HorizontalSyncActive*
- *uint32\_t HorizontalBackPorch*
- *uint32\_t HorizontalLine*

- ***uint32\_t VerticalSyncActive***
- ***uint32\_t VerticalBackPorch***
- ***uint32\_t VerticalFrontPorch***
- ***uint32\_t VerticalActive***
- ***uint32\_t LPCommandEnable***
- ***uint32\_t LPLargestPacketSize***
- ***uint32\_t LPVACTLargestPacketSize***
- ***uint32\_t LPHorizontalFrontPorchEnable***
- ***uint32\_t LPHorizontalBackPorchEnable***
- ***uint32\_t LPVerticalActiveEnable***
- ***uint32\_t LPVerticalFrontPorchEnable***
- ***uint32\_t LPVerticalBackPorchEnable***
- ***uint32\_t LPVerticalSyncActiveEnable***
- ***uint32\_t FrameBTAcknowledgeEnable***

**Field Documentation**

- ***uint32\_t DSI\_VidCfgTypeDef::VirtualChannelID***  
Virtual channel ID
- ***uint32\_t DSI\_VidCfgTypeDef::ColorCoding***  
Color coding for LTDC interface This parameter can be any value of [DSI\\_Color\\_Coding](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LooselyPacked***  
Enable or disable loosely packed stream (needed only when using 18-bit configuration). This parameter can be any value of [DSI\\_LooselyPacked](#)
- ***uint32\_t DSI\_VidCfgTypeDef::Mode***  
Video mode type This parameter can be any value of [DSI\\_Video\\_Mode\\_Type](#)
- ***uint32\_t DSI\_VidCfgTypeDef::PacketSize***  
Video packet size
- ***uint32\_t DSI\_VidCfgTypeDef::NumberOfChunks***  
Number of chunks
- ***uint32\_t DSI\_VidCfgTypeDef::NullPacketSize***  
Null packet size
- ***uint32\_t DSI\_VidCfgTypeDef::HSPolarity***  
HSYNC pin polarity This parameter can be any value of [DSI\\_HSYNC\\_Polarity](#)
- ***uint32\_t DSI\_VidCfgTypeDef::VSPolarity***  
VSYNC pin polarity This parameter can be any value of [DSI\\_VSYNC\\_Active\\_Polarity](#)
- ***uint32\_t DSI\_VidCfgTypeDef::DEPolarity***  
Data Enable pin polarity This parameter can be any value of [DSI\\_DATA\\_ENABLE\\_Polarity](#)
- ***uint32\_t DSI\_VidCfgTypeDef::HorizontalSyncActive***  
Horizontal synchronism active duration (in lane byte clock cycles)
- ***uint32\_t DSI\_VidCfgTypeDef::HorizontalBackPorch***  
Horizontal back-porch duration (in lane byte clock cycles)
- ***uint32\_t DSI\_VidCfgTypeDef::HorizontalLine***  
Horizontal line duration (in lane byte clock cycles)
- ***uint32\_t DSI\_VidCfgTypeDef::VerticalSyncActive***  
Vertical synchronism active duration
- ***uint32\_t DSI\_VidCfgTypeDef::VerticalBackPorch***  
Vertical back-porch duration
- ***uint32\_t DSI\_VidCfgTypeDef::VerticalFrontPorch***  
Vertical front-porch duration
- ***uint32\_t DSI\_VidCfgTypeDef::VerticalActive***  
Vertical active duration



- ***uint32\_t DSI\_VidCfgTypeDef::LPCommandEnable***  
Low-power command enable This parameter can be any value of [DSI\\_LP\\_Command](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LPLargestPacketSize***  
The size, in bytes, of the low power largest packet that can fit in a line during VSA, VBP and VFP regions
- ***uint32\_t DSI\_VidCfgTypeDef::LPVACTLargestPacketSize***  
The size, in bytes, of the low power largest packet that can fit in a line during VACT region
- ***uint32\_t DSI\_VidCfgTypeDef::LPHorizontalFrontPorchEnable***  
Low-power horizontal front-porch enable This parameter can be any value of [DSI\\_LP\\_HFP](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LPHorizontalBackPorchEnable***  
Low-power horizontal back-porch enable This parameter can be any value of [DSI\\_LP\\_HBP](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LPVerticalActiveEnable***  
Low-power vertical active enable This parameter can be any value of [DSI\\_LP\\_VACT](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LPVerticalFrontPorchEnable***  
Low-power vertical front-porch enable This parameter can be any value of [DSI\\_LP\\_VFP](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LPVerticalBackPorchEnable***  
Low-power vertical back-porch enable This parameter can be any value of [DSI\\_LP\\_VBP](#)
- ***uint32\_t DSI\_VidCfgTypeDef::LPVerticalSyncActiveEnable***  
Low-power vertical sync active enable This parameter can be any value of [DSI\\_LP\\_VSYNC](#)
- ***uint32\_t DSI\_VidCfgTypeDef::FrameBTAAcknowledgeEnable***  
Frame bus-turn-around acknowledge enable This parameter can be any value of [DSI\\_FBTA\\_acknowledge](#)

### 23.1.4

#### DSI\_CmdCfgTypeDef

*DSI\_CmdCfgTypeDef* is defined in the `stm32f4xx_hal_dsi.h`

##### Data Fields

- ***uint32\_t VirtualChannelID***
- ***uint32\_t ColorCoding***
- ***uint32\_t CommandSize***
- ***uint32\_t TearingEffectSource***
- ***uint32\_t TearingEffectPolarity***
- ***uint32\_t HSPolarity***
- ***uint32\_t VSPolarity***
- ***uint32\_t DEPolarity***
- ***uint32\_t VSyncPol***
- ***uint32\_t AutomaticRefresh***
- ***uint32\_t TEAcknowledgeRequest***

##### Field Documentation

- ***uint32\_t DSI\_CmdCfgTypeDef::VirtualChannelID***  
Virtual channel ID
- ***uint32\_t DSI\_CmdCfgTypeDef::ColorCoding***  
Color coding for LTDC interface This parameter can be any value of [DSI\\_Color\\_Coding](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::CommandSize***  
Maximum allowed size for an LTDC write memory command, measured in pixels. This parameter can be any value between 0x00 and 0xFFFFU
- ***uint32\_t DSI\_CmdCfgTypeDef::TearingEffectSource***  
Tearing effect source This parameter can be any value of [DSI\\_TearingEffectSource](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::TearingEffectPolarity***  
Tearing effect pin polarity This parameter can be any value of [DSI\\_TearingEffectPolarity](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::HSPolarity***  
HSYNC pin polarity This parameter can be any value of [DSI\\_HSYNC\\_Polarity](#)

- ***uint32\_t DSI\_CmdCfgTypeDef::VSPolarity***  
VSYNC pin polarity This parameter can be any value of [DSI\\_VSYNC\\_Active\\_Polarity](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::DEPolarity***  
Data Enable pin polarity This parameter can be any value of [DSI\\_DATA\\_ENABLE\\_Polarity](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::VSyncPol***  
VSync edge on which the LTDC is halted This parameter can be any value of [DSI\\_Vsync\\_Polarity](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::AutomaticRefresh***  
Automatic refresh mode This parameter can be any value of [DSI\\_AutomaticRefresh](#)
- ***uint32\_t DSI\_CmdCfgTypeDef::TEAcknowledgeRequest***  
Tearing Effect Acknowledge Request Enable This parameter can be any value of [DSI\\_TE\\_AcknowledgeRequest](#)

### 23.1.5

#### DSI\_LPCmdTypeDef

*DSI\_LPCmdTypeDef* is defined in the `stm32f4xx_hal_dsi.h`

##### Data Fields

- ***uint32\_t LPGenShortWriteNoP***
- ***uint32\_t LPGenShortWriteOneP***
- ***uint32\_t LPGenShortWriteTwoP***
- ***uint32\_t LPGenShortReadNoP***
- ***uint32\_t LPGenShortReadOneP***
- ***uint32\_t LPGenShortReadTwoP***
- ***uint32\_t LPGenLongWrite***
- ***uint32\_t LPDcsShortWriteNoP***
- ***uint32\_t LPDcsShortWriteOneP***
- ***uint32\_t LPDcsShortReadNoP***
- ***uint32\_t LPDcsLongWrite***
- ***uint32\_t LPMaxReadPacket***
- ***uint32\_t AcknowledgeRequest***

##### Field Documentation

- ***uint32\_t DSI\_LPCmdTypeDef::LPGenShortWriteNoP***  
Generic Short Write Zero parameters Transmission This parameter can be any value of [DSI\\_LP\\_LPGenShortWriteNoP](#)
- ***uint32\_t DSI\_LPCmdTypeDef::LPGenShortWriteOneP***  
Generic Short Write One parameter Transmission This parameter can be any value of [DSI\\_LP\\_LPGenShortWriteOneP](#)
- ***uint32\_t DSI\_LPCmdTypeDef::LPGenShortWriteTwoP***  
Generic Short Write Two parameters Transmission This parameter can be any value of [DSI\\_LP\\_LPGenShortWriteTwoP](#)
- ***uint32\_t DSI\_LPCmdTypeDef::LPGenShortReadNoP***  
Generic Short Read Zero parameters Transmission This parameter can be any value of [DSI\\_LP\\_LPGenShortReadNoP](#)
- ***uint32\_t DSI\_LPCmdTypeDef::LPGenShortReadOneP***  
Generic Short Read One parameter Transmission This parameter can be any value of [DSI\\_LP\\_LPGenShortReadOneP](#)
- ***uint32\_t DSI\_LPCmdTypeDef::LPGenShortReadTwoP***  
Generic Short Read Two parameters Transmission This parameter can be any value of [DSI\\_LP\\_LPGenShortReadTwoP](#)
- ***uint32\_t DSI\_LPCmdTypeDef::LPGenLongWrite***  
Generic Long Write Transmission This parameter can be any value of [DSI\\_LP\\_LPGenLongWrite](#)

- **`uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteNoP`**  
DCS Short Write Zero parameters Transmission This parameter can be any value of [DSI\\_LP\\_LPDCsShortWriteNoP](#)
- **`uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteOneP`**  
DCS Short Write One parameter Transmission This parameter can be any value of [DSI\\_LP\\_LPDCsShortWriteOneP](#)
- **`uint32_t DSI_LPCmdTypeDef::LPDcsShortReadNoP`**  
DCS Short Read Zero parameters Transmission This parameter can be any value of [DSI\\_LP\\_LPDCsShortReadNoP](#)
- **`uint32_t DSI_LPCmdTypeDef::LPDcsLongWrite`**  
DCS Long Write Transmission This parameter can be any value of [DSI\\_LP\\_LPDCsLongWrite](#)
- **`uint32_t DSI_LPCmdTypeDef::LPMaxReadPacket`**  
Maximum Read Packet Size Transmission This parameter can be any value of [DSI\\_LP\\_LPMaxReadPacket](#)
- **`uint32_t DSI_LPCmdTypeDef::AcknowledgeRequest`**  
Acknowledge Request Enable This parameter can be any value of [DSI\\_AcknowledgeRequest](#)

### 23.1.6

#### DSI\_PHY\_TimerTypeDef

**`DSI_PHY_TimerTypeDef`** is defined in the `stm32f4xx_hal_dsi.h`

##### Data Fields

- **`uint32_t ClockLaneHS2LPTime`**
- **`uint32_t ClockLaneLP2HSTime`**
- **`uint32_t DataLaneHS2LPTime`**
- **`uint32_t DataLaneLP2HSTime`**
- **`uint32_t DataLaneMaxReadTime`**
- **`uint32_t StopWaitTime`**

##### Field Documentation

- **`uint32_t DSI_PHY_TimerTypeDef::ClockLaneHS2LPTime`**  
The maximum time that the D-PHY clock lane takes to go from high-speed to low-power transmission
- **`uint32_t DSI_PHY_TimerTypeDef::ClockLaneLP2HSTime`**  
The maximum time that the D-PHY clock lane takes to go from low-power to high-speed transmission
- **`uint32_t DSI_PHY_TimerTypeDef::DataLaneHS2LPTime`**  
The maximum time that the D-PHY data lanes takes to go from high-speed to low-power transmission
- **`uint32_t DSI_PHY_TimerTypeDef::DataLaneLP2HSTime`**  
The maximum time that the D-PHY data lanes takes to go from low-power to high-speed transmission
- **`uint32_t DSI_PHY_TimerTypeDef::DataLaneMaxReadTime`**  
The maximum time required to perform a read command
- **`uint32_t DSI_PHY_TimerTypeDef::StopWaitTime`**  
The minimum wait period to request a High-Speed transmission after the Stop state

### 23.1.7

#### DSI\_HOST\_TimeoutTypeDef

**`DSI_HOST_TimeoutTypeDef`** is defined in the `stm32f4xx_hal_dsi.h`

##### Data Fields

- **`uint32_t TimeoutCkdiv`**
- **`uint32_t HighSpeedTransmissionTimeout`**
- **`uint32_t LowPowerReceptionTimeout`**
- **`uint32_t HighSpeedReadTimeout`**
- **`uint32_t LowPowerReadTimeout`**
- **`uint32_t HighSpeedWriteTimeout`**
- **`uint32_t HighSpeedWritePrespMode`**
- **`uint32_t LowPowerWriteTimeout`**
- **`uint32_t BTATimeout`**

**Field Documentation**

- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::TimeoutCkdiv***  
Time-out clock division
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::HighSpeedTransmissionTimeout***  
High-speed transmission time-out
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::LowPowerReceptionTimeout***  
Low-power reception time-out
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::HighSpeedReadTimeout***  
High-speed read time-out
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::LowPowerReadTimeout***  
Low-power read time-out
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::HighSpeedWriteTimeout***  
High-speed write time-out
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::HighSpeedWritePrespMode***  
High-speed write presp mode This parameter can be any value of *DSI\_HS\_PrespMode*
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::LowPowerWriteTimeout***  
Low-speed write time-out
- ***uint32\_t DSI\_HOST\_TimeoutTypeDef::BTATimeout***  
BTA time-out

**23.1.8**
**DSI\_HandleTypeDef**

*DSI\_HandleTypeDef* is defined in the stm32f4xx\_hal\_dsi.h

**Data Fields**

- ***DSI\_TypeDef \* Instance***
- ***DSI\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_DSI\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***
- ***uint32\_t ErrorMsk***

**Field Documentation**

- ***DSI\_TypeDef\* DSI\_HandleTypeDef::Instance***  
Register base address
- ***DSI\_InitTypeDef DSI\_HandleTypeDef::Init***  
DSI required parameters
- ***HAL\_LockTypeDef DSI\_HandleTypeDef::Lock***  
DSI peripheral status
- ***\_\_IO HAL\_DSI\_StateTypeDef DSI\_HandleTypeDef::State***  
DSI communication state
- ***\_\_IO uint32\_t DSI\_HandleTypeDef::ErrorCode***  
DSI Error code
- ***uint32\_t DSI\_HandleTypeDef::ErrorMsk***  
DSI Error monitoring mask

**23.2**
**DSI Firmware driver API description**

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

**23.2.1**
**How to use this driver**

The DSI HAL driver can be used as follows:

1. Declare a *DSI\_HandleTypeDef* handle structure, for example: *DSI\_HandleTypeDef hdsi*;

2. Initialize the DSI low level resources by implementing the HAL\_DSI\_Msplnit() API:
  - a. Enable the DSI interface clock
  - b. NVIC configuration if you need to use interrupt process
    - Configure the DSI interrupt priority
    - Enable the NVIC DSI IRQ Channel
3. Initialize the DSI Host peripheral, the required PLL parameters, number of lances and TX Escape clock divider by calling the HAL\_DSI\_Init() API which calls HAL\_DSI\_Msplnit().

### Configuration

1. Use HAL\_DSI\_ConfigAdaptedCommandMode() function to configure the DSI host in adapted command mode.
2. When operating in video mode , use HAL\_DSI\_ConfigVideoMode() to configure the DSI host.
3. Function HAL\_DSI\_ConfigCommand() is used to configure the DSI commands behavior in low power mode.
4. To configure the DSI PHY timings parameters, use function HAL\_DSI\_ConfigPhyTimer().
5. The DSI Host can be started/stopped using respectively functions HAL\_DSI\_Start() and HAL\_DSI\_Stop(). Functions HAL\_DSI\_ShortWrite(), HAL\_DSI\_LongWrite() and HAL\_DSI\_Read() allows respectively to write DSI short packets, long packets and to read DSI packets.
6. The DSI Host Offers two Low power modes :
  - Low Power Mode on data lanes only: Only DSI data lanes are shut down. It is possible to enter/exit from this mode using respectively functions HAL\_DSI\_EnterULPMData() and HAL\_DSI\_ExitULPMData()
  - Low Power Mode on data and clock lanes : All DSI lanes are shut down including data and clock lanes. It is possible to enter/exit from this mode using respectively functions HAL\_DSI\_EnterULPM() and HAL\_DSI\_ExitULPM()
7. To control DSI state you can use the following function: HAL\_DSI\_GetState()

### Error management

1. User can select the DSI errors to be reported/monitored using function HAL\_DSI\_ConfigErrorMonitor() When an error occurs, the callback HAL\_DSI\_ErrorCallback() is asserted and then user can retrieve the error code by calling function HAL\_DSI\_GetError()

### DSI HAL driver macros list

Below the list of most used macros in DSI HAL driver.

- `__HAL_DSI_ENABLE`: Enable the DSI Host.
- `__HAL_DSI_DISABLE`: Disable the DSI Host.
- `__HAL_DSI_WRAPPER_ENABLE`: Enables the DSI wrapper.
- `__HAL_DSI_WRAPPER_DISABLE`: Disable the DSI wrapper.
- `__HAL_DSI_PLL_ENABLE`: Enables the DSI PLL.
- `__HAL_DSI_PLL_DISABLE`: Disables the DSI PLL.
- `__HAL_DSI_REG_ENABLE`: Enables the DSI regulator.
- `__HAL_DSI_REG_DISABLE`: Disables the DSI regulator.
- `__HAL_DSI_GET_FLAG`: Get the DSI pending flags.
- `__HAL_DSI_CLEAR_FLAG`: Clears the DSI pending flags.
- `__HAL_DSI_ENABLE_IT`: Enables the specified DSI interrupts.
- `__HAL_DSI_DISABLE_IT`: Disables the specified DSI interrupts.
- `__HAL_DSI_GET_IT_SOURCE`: Checks whether the specified DSI interrupt source is enabled or not.

*Note:* You can refer to the DSI HAL driver header file for more useful macros

### Callback registration

The compilation define `USE_HAL_DSI_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use Function `HAL_DSI_RegisterCallback()` to register a callback.

Function `HAL_DSI_RegisterCallback()` allows to register following callbacks:

- `TearingEffectCallback` : DSI Tearing Effect Callback.
- `EndOfRefreshCallback` : DSI End Of Refresh Callback.
- `ErrorCallback` : DSI Error Callback
- `MspInitCallback` : DSI MspInit.
- `MspDeInitCallback` : DSI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function `HAL_DSI_UnRegisterCallback()` to reset a callback to the default weak function. `HAL_DSI_UnRegisterCallback` takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- `TearingEffectCallback` : DSI Tearing Effect Callback.
- `EndOfRefreshCallback` : DSI End Of Refresh Callback.
- `ErrorCallback` : DSI Error Callback
- `MspInitCallback` : DSI MspInit.
- `MspDeInitCallback` : DSI MspDeInit.

By default, after the `HAL_DSI_Init` and when the state is `HAL_DSI_STATE_RESET` all callbacks are set to the corresponding weak functions: examples `HAL_DSI_TearingEffectCallback()`, `HAL_DSI_EndOfRefreshCallback()`. Exception done for `MspInit` and `MspDeInit` functions that are respectively reset to the legacy weak (surcharged) functions in the `HAL_DSI_Init()` and `HAL_DSI_DeInit()` only when these callbacks are null (not registered beforehand). If not, `MspInit` or `MspDeInit` are not null, the `HAL_DSI_Init()` and `HAL_DSI_DeInit()` keep and use the user `MspInit/MspDeInit` callbacks (registered beforehand).

Callbacks can be registered/unregistered in `HAL_DSI_STATE_READY` state only. Exception done `MspInit/MspDeInit` that can be registered/unregistered in `HAL_DSI_STATE_READY` or `HAL_DSI_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 `HAL_DSI_RegisterCallback()` before calling `HAL_DSI_DeInit()` or `HAL_DSI_Init()` function.

When The compilation define `USE_HAL_DSI_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.

### 23.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DSI
- De-initialize the DSI

This section contains the following APIs:

- [\*\*\*HAL\\_DSI\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_DeInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_ConfigErrorMonitor\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_MspInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_MspDeInit\(\)\*\*\*](#)

### 23.2.3 IO operation functions

This section provides function allowing to:

- Handle DSI interrupt request

This section contains the following APIs:

- [\*\*\*HAL\\_DSI\\_IRQHandler\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_TearingEffectCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_EndOfRefreshCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_DSI\\_ErrorCallback\(\)\*\*\*](#)

### 23.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the Generic interface read-back Virtual Channel ID
- Select video mode and configure the corresponding parameters
- Configure command transmission mode: High-speed or Low-power
- Configure the flow control
- Configure the DSI PHY timer
- Configure the DSI HOST timeout
- Configure the DSI HOST timeout
- Start/Stop the DSI module
- Refresh the display in command mode
- Controls the display color mode in Video mode
- Control the display shutdown in Video mode
- write short DCS or short Generic command
- write long DCS or long Generic command
- Read command (DCS or generic)
- Enter/Exit the Ultra Low Power Mode on data only (D-PHY PLL running)
- Enter/Exit the Ultra Low Power Mode on data only and clock (D-PHY PLL turned off)
- Start/Stop test pattern generation
- Slew-Rate And Delay Tuning
- Low-Power Reception Filter Tuning
- Activate an additional current path on all lanes to meet the SDDTx parameter
- Custom lane pins configuration
- Set custom timing for the PHY
- Force the Clock/Data Lane in TX Stop Mode
- Force LP Receiver in Low-Power Mode
- Force Data Lanes in RX Mode after a BTA
- Enable a pull-down on the lanes to prevent from floating states when unused
- Switch off the contention detection on data lanes

This section contains the following APIs:

- ***HAL\_DSI\_SetGenericVCID()***
- ***HAL\_DSI\_ConfigVideoMode()***
- ***HAL\_DSI\_ConfigAdaptedCommandMode()***
- ***HAL\_DSI\_ConfigCommand()***
- ***HAL\_DSI\_ConfigFlowControl()***
- ***HAL\_DSI\_ConfigPhyTimer()***
- ***HAL\_DSI\_ConfigHostTimeouts()***
- ***HAL\_DSI\_Start()***
- ***HAL\_DSI\_Stop()***
- ***HAL\_DSI\_Refresh()***
- ***HAL\_DSI\_ColorMode()***
- ***HAL\_DSI\_Shutdown()***
- ***HAL\_DSI\_ShortWrite()***
- ***HAL\_DSI\_LongWrite()***
- ***HAL\_DSI\_Read()***
- ***HAL\_DSI\_EnterULPMData()***
- ***HAL\_DSI\_ExitULPMData()***
- ***HAL\_DSI\_EnterULPM()***
- ***HAL\_DSI\_ExitULPM()***
- ***HAL\_DSI\_PatternGeneratorStart()***
- ***HAL\_DSI\_PatternGeneratorStop()***
- ***HAL\_DSI\_SetSlewRateAndDelayTuning()***

- *HAL\_DSI\_SetLowPowerRXFilter()*
- *HAL\_DSI\_SetSDD()*
- *HAL\_DSI\_SetLanePinsConfiguration()*
- *HAL\_DSI\_SetPHYTimings()*
- *HAL\_DSI\_ForceTXStopMode()*
- *HAL\_DSI\_ForceRXLowPower()*
- *HAL\_DSI\_ForceDataLanesInRX()*
- *HAL\_DSI\_SetPullDown()*
- *HAL\_DSI\_SetContentionDetectionOff()*

### 23.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DSI state.
- Get error code.

This section contains the following APIs:

- *HAL\_DSI\_GetState()*
- *HAL\_DSI\_GetError()*

### 23.2.6 Detailed description of functions

#### HAL\_DSI\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_DSI\_Init (DSI\_HandleTypeDef \* hdsi, DSI\_PLLInitTypeDef \* PLLInit)**

##### Function description

Initializes the DSI according to the specified parameters in the DSI\_InitTypeDef and create the associated handle.

##### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **PLLInit**: pointer to a DSI\_PLLInitTypeDef structure that contains the PLL Clock structure definition for the DSI.

##### Return values

- **HAL**: status

#### HAL\_DSI\_DeInit

##### Function name

**HAL\_StatusTypeDef HAL\_DSI\_DeInit (DSI\_HandleTypeDef \* hdsi)**

##### Function description

De-initializes the DSI peripheral registers to their default reset values.

##### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

##### Return values

- **HAL**: status

#### HAL\_DSI\_MspInit

##### Function name

**void HAL\_DSI\_MspInit (DSI\_HandleTypeDef \* hdsi)**



### Function description

Initializes the DSI MSP.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

### Return values

- **None**:

**HAL\_DSI\_MspDeInit**

### Function name

**void HAL\_DSI\_MspDeInit (DSI\_HandleTypeDef \* hdsi)**

### Function description

De-initializes the DSI MSP.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

### Return values

- **None**:

**HAL\_DSI\_IRQHandler**

### Function name

**void HAL\_DSI\_IRQHandler (DSI\_HandleTypeDef \* hdsi)**

### Function description

Handles DSI interrupt request.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

### Return values

- **HAL**: status

**HAL\_DSI\_TearingEffectCallback**

### Function name

**void HAL\_DSI\_TearingEffectCallback (DSI\_HandleTypeDef \* hdsi)**

### Function description

Tearing Effect DSI callback.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

### Return values

- **None**:

**HAL\_DSI\_EndOfRefreshCallback**

### Function name

**void HAL\_DSI\_EndOfRefreshCallback (DSI\_HandleTypeDef \* hdsi)**

### Function description

End of Refresh DSI callback.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **None**:

**HAL\_DSI\_ErrorCallback**

**Function name**

**void HAL\_DSI\_ErrorCallback (DSI\_HandleTypeDef \* hdsi)**

**Function description**

Operation Error DSI callback.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **None**:

**HAL\_DSI\_SetGenericVCID**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_SetGenericVCID (DSI\_HandleTypeDef \* hdsi, uint32\_t VirtualChannelID)**

**Function description**

Configure the Generic interface read-back Virtual Channel ID.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **VirtualChannelID**: Virtual channel ID

**Return values**

- **HAL**: status

**HAL\_DSI\_ConfigVideoMode**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_ConfigVideoMode (DSI\_HandleTypeDef \* hdsi, DSI\_VidCfgTypeDef \* VidCfg)**

**Function description**

Select video mode and configure the corresponding parameters.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **VidCfg**: pointer to a DSI\_VidCfgTypeDef structure that contains the DSI video mode configuration parameters

**Return values**

- **HAL**: status

**HAL\_DSI\_ConfigAdaptedCommandMode**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_ConfigAdaptedCommandMode (DSI\_HandleTypeDef \* hdsi, DSI\_CmdCfgTypeDef \* CmdCfg)**

### Function description

Select adapted command mode and configure the corresponding parameters.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **CmdCfg**: pointer to a DSI\_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters

### Return values

- **HAL**: status

### HAL\_DSI\_ConfigCommand

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ConfigCommand (DSI\_HandleTypeDef \* hdsi, DSI\_LPCmdTypeDef \* LPCmd)**

### Function description

Configure command transmission mode: High-speed or Low-power and enable/disable acknowledge request after packet transmission.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **LPCmd**: pointer to a DSI\_LPCmdTypeDef structure that contains the DSI command transmission mode configuration parameters

### Return values

- **HAL**: status

### HAL\_DSI\_ConfigFlowControl

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ConfigFlowControl (DSI\_HandleTypeDef \* hdsi, uint32\_t FlowControl)**

### Function description

Configure the flow control parameters.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **FlowControl**: flow control feature(s) to be enabled. This parameter can be any combination of
  - DSI\_FlowControl.

### Return values

- **HAL**: status

### HAL\_DSI\_ConfigPhyTimer

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ConfigPhyTimer (DSI\_HandleTypeDef \* hdsi, DSI\_PHY\_TimerTypeDef \* PhyTimers)**

### Function description

Configure the DSI PHY timer parameters.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **PhyTimers**: DSI\_PHY\_TimerTypeDef structure that contains the DSI PHY timing parameters

#### Return values

- **HAL:** status

#### HAL\_DSI\_ConfigHostTimeouts

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ConfigHostTimeouts (DSI\_HandleTypeDef \* hdsi, DSI\_HOST\_TimeoutTypeDef \* HostTimeouts)**

#### Function description

Configure the DSI HOST timeout parameters.

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **HostTimeouts:** DSI\_HOST\_TimeoutTypeDef structure that contains the DSI host timeout parameters

#### Return values

- **HAL:** status

#### HAL\_DSI\_Start

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_Start (DSI\_HandleTypeDef \* hdsi)**

#### Function description

Start the DSI module.

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

#### Return values

- **HAL:** status

#### HAL\_DSI\_Stop

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_Stop (DSI\_HandleTypeDef \* hdsi)**

#### Function description

Stop the DSI module.

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

#### Return values

- **HAL:** status

#### HAL\_DSI\_Refresh

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_Refresh (DSI\_HandleTypeDef \* hdsi)**

#### Function description

Refresh the display in command mode.

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

#### Return values

- **HAL:** status

#### **HAL\_DSI\_ColorMode**

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ColorMode (DSI\_HandleTypeDef \* hdsi, uint32\_t ColorMode)**

#### Function description

Controls the display color mode in Video mode.

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **ColorMode:** Color mode (full or 8-colors). This parameter can be any value of
  - DSI\_Color\_Mode

#### Return values

- **HAL:** status

#### **HAL\_DSI\_Shutdown**

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_Shutdown (DSI\_HandleTypeDef \* hdsi, uint32\_t Shutdown)**

#### Function description

Control the display shutdown in Video mode.

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **Shutdown:** Shut-down (Display-ON or Display-OFF). This parameter can be any value of
  - DSI\_ShutDown

#### Return values

- **HAL:** status

#### **HAL\_DSI\_ShortWrite**

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ShortWrite (DSI\_HandleTypeDef \* hdsi, uint32\_t ChannelID, uint32\_t Mode, uint32\_t Param1, uint32\_t Param2)**

#### Function description

write short DCS or short Generic command

#### Parameters

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelID:** Virtual channel ID.
- **Mode:** DSI short packet data type. This parameter can be any value of
  - DSI\_SHORT\_WRITE\_PKT\_Data\_Type.
- **Param1:** DSC command or first generic parameter. This parameter can be any value of
  - DSI\_DCS\_Command or a generic command code.
- **Param2:** DSC parameter or second generic parameter.

#### Return values

- **HAL:** status

## HAL\_DSI\_LongWrite

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_LongWrite (DSI\_HandleTypeDef \* hdsi, uint32\_t ChannelID, uint32\_t Mode, uint32\_t NbParams, uint32\_t Param1, uint8\_t \* ParametersTable)**

### Function description

write long DCS or long Generic command

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelID**: Virtual channel ID.
- **Mode**: DSI long packet data type. This parameter can be any value of
  - DSI\_LONG\_WRITE\_PKT\_Data\_Type.
- **NbParams**: Number of parameters.
- **Param1**: DSC command or first generic parameter. This parameter can be any value of
  - DSI\_DCS\_Command or a generic command code
- **ParametersTable**: Pointer to parameter values table.

### Return values

- **HAL**: status

## HAL\_DSI\_Read

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_Read (DSI\_HandleTypeDef \* hdsi, uint32\_t ChannelNbr, uint8\_t \* Array, uint32\_t Size, uint32\_t Mode, uint32\_t DCSCmd, uint8\_t \* ParametersTable)**

### Function description

Read command (DCS or generic)

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelNbr**: Virtual channel ID
- **Array**: pointer to a buffer to store the payload of a read back operation.
- **Size**: Data size to be read (in byte).
- **Mode**: DSI read packet data type. This parameter can be any value of
  - DSI\_SHORT\_READ\_PKT\_Data\_Type.
- **DCSCmd**: DCS get/read command.
- **ParametersTable**: Pointer to parameter values table.

### Return values

- **HAL**: status

## HAL\_DSI\_EnterULPMData

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_EnterULPMData (DSI\_HandleTypeDef \* hdsi)**

### Function description

Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **HAL:** status

**HAL\_DSI\_ExitULPMData**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_ExitULPMData (DSI\_HandleTypeDef \* hdsi)**

**Function description**

Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)

**Parameters**

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **HAL:** status

**HAL\_DSI\_EnterULPM**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_EnterULPM (DSI\_HandleTypeDef \* hdsi)**

**Function description**

Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)

**Parameters**

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **HAL:** status

**HAL\_DSI\_ExitULPM**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_ExitULPM (DSI\_HandleTypeDef \* hdsi)**

**Function description**

Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)

**Parameters**

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **HAL:** status

**HAL\_DSI\_PatternGeneratorStart**

**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_PatternGeneratorStart (DSI\_HandleTypeDef \* hdsi, uint32\_t Mode, uint32\_t Orientation)**

**Function description**

Start test pattern generation.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **Mode**: Pattern generator mode This parameter can be one of the following values: 0 : Color bars (horizontal or vertical) 1 : BER pattern (vertical only)
- **Orientation**: Pattern generator orientation This parameter can be one of the following values: 0 : Vertical color bars 1 : Horizontal color bars

### Return values

- **HAL**: status

### HAL\_DSI\_PatternGeneratorStop

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_PatternGeneratorStop (DSI\_HandleTypeDef \* hdsi)**

### Function description

Stop test pattern generation.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

### Return values

- **HAL**: status

### HAL\_DSI\_SetSlewRateAndDelayTuning

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_SetSlewRateAndDelayTuning (DSI\_HandleTypeDef \* hdsi, uint32\_t CommDelay, uint32\_t Lane, uint32\_t Value)**

### Function description

Set Slew-Rate And Delay Tuning.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **CommDelay**: Communication delay to be adjusted. This parameter can be any value of
  - DSI\_Communication\_Delay
- **Lane**: select between clock or data lanes. This parameter can be any value of
  - DSI\_Lane\_Group
- **Value**: Custom value of the slew-rate or delay

### Return values

- **HAL**: status

### HAL\_DSI\_SetLowPowerRXFilter

### Function name

**HAL\_StatusTypeDef HAL\_DSI\_SetLowPowerRXFilter (DSI\_HandleTypeDef \* hdsi, uint32\_t Frequency)**

### Function description

Low-Power Reception Filter Tuning.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **Frequency**: cutoff frequency of low-pass filter at the input of LPRX



**Return values**

- **HAL:** status

**HAL\_DSI\_SetSDD**
**Function name**
**HAL\_StatusTypeDef HAL\_DSI\_SetSDD (DSI\_HandleTypeDef \* hdsi, FunctionalState State)**
**Function description**

Activate an additional current path on all lanes to meet the SDDTx parameter defined in the MIPI D-PHY specification.

**Parameters**

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **State:** ENABLE or DISABLE

**Return values**

- **HAL:** status

**HAL\_DSI\_SetLanePinsConfiguration**
**Function name**
**HAL\_StatusTypeDef HAL\_DSI\_SetLanePinsConfiguration (DSI\_HandleTypeDef \* hdsi, uint32\_t CustomLane, uint32\_t Lane, FunctionalState State)**
**Function description**

Custom lane pins configuration.

**Parameters**

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **CustomLane:** Function to be applied on selected lane. This parameter can be any value of
  - DSI\_CustomLane
- **Lane:** select between clock or data lane 0 or data lane 1. This parameter can be any value of
  - DSI\_Lane\_Select
- **State:** ENABLE or DISABLE

**Return values**

- **HAL:** status

**HAL\_DSI\_SetPHYTimings**
**Function name**
**HAL\_StatusTypeDef HAL\_DSI\_SetPHYTimings (DSI\_HandleTypeDef \* hdsi, uint32\_t Timing, FunctionalState State, uint32\_t Value)**
**Function description**

Set custom timing for the PHY.

**Parameters**

- **hdsi:** pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **Timing:** PHY timing to be adjusted. This parameter can be any value of
  - DSI\_PHY\_Timing
- **State:** ENABLE or DISABLE
- **Value:** Custom value of the timing

**Return values**

- **HAL:** status

### HAL\_DSI\_ForceTXStopMode

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ForceTXStopMode (DSI\_HandleTypeDef \* hdsi, uint32\_t Lane, FunctionalState State)**

#### Function description

Force the Clock/Data Lane in TX Stop Mode.

#### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **Lane**: select between clock or data lanes. This parameter can be any value of
  - DSI\_Lane\_Group
- **State**: ENABLE or DISABLE

#### Return values

- **HAL**: status

### HAL\_DSI\_ForceRXLowPower

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ForceRXLowPower (DSI\_HandleTypeDef \* hdsi, FunctionalState State)**

#### Function description

Force LP Receiver in Low-Power Mode.

#### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **State**: ENABLE or DISABLE

#### Return values

- **HAL**: status

### HAL\_DSI\_ForceDataLanesInRX

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_ForceDataLanesInRX (DSI\_HandleTypeDef \* hdsi, FunctionalState State)**

#### Function description

Force Data Lanes in RX Mode after a BTA.

#### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **State**: ENABLE or DISABLE

#### Return values

- **HAL**: status

### HAL\_DSI\_SetPullDown

#### Function name

**HAL\_StatusTypeDef HAL\_DSI\_SetPullDown (DSI\_HandleTypeDef \* hdsi, FunctionalState State)**

#### Function description

Enable a pull-down on the lanes to prevent from floating states when unused.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **State**: ENABLE or DISABLE

**Return values**

- **HAL**: status

**HAL\_DSI\_SetContentionDetectionOff**
**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_SetContentionDetectionOff (DSI\_HandleTypeDef \* hdsi, FunctionalState State)**

**Function description**

Switch off the contention detection on data lanes.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **State**: ENABLE or DISABLE

**Return values**

- **HAL**: status

**HAL\_DSI\_GetError**
**Function name**

**uint32\_t HAL\_DSI\_GetError (DSI\_HandleTypeDef \* hdsi)**

**Function description**

Return the DSI error code.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

**Return values**

- **DSI**: Error Code

**HAL\_DSI\_ConfigErrorMonitor**
**Function name**

**HAL\_StatusTypeDef HAL\_DSI\_ConfigErrorMonitor (DSI\_HandleTypeDef \* hdsi, uint32\_t ActiveErrors)**

**Function description**

Enable the error monitor flags.

**Parameters**

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.
- **ActiveErrors**: indicates which error interrupts will be enabled. This parameter can be any combination of
  - DSI\_Error\_Data\_Type.

**Return values**

- **HAL**: status

**HAL\_DSI\_GetState**
**Function name**

**HAL\_DSI\_StateTypeDef HAL\_DSI\_GetState (DSI\_HandleTypeDef \* hdsi)**

### Function description

Return the DSI state.

### Parameters

- **hdsi**: pointer to a DSI\_HandleTypeDef structure that contains the configuration information for the DSI.

### Return values

- **HAL**: state

## 23.3 DSI Firmware driver defines

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

### 23.3.1 DSI

DSI

*DSI Acknowledge Request*

**DSI\_ACKNOWLEDGE\_DISABLE**

**DSI\_ACKNOWLEDGE\_ENABLE**

*DSI Automatic Refresh*

**DSI\_AR\_DISABLE**

**DSI\_AR\_ENABLE**

*DSI Automatic Clk Lane Control*

**DSI\_AUTO\_CLK\_LANE\_CTRL\_DISABLE**

**DSI\_AUTO\_CLK\_LANE\_CTRL\_ENABLE**

*DSI Color Coding*

**DSI\_RGB565**

The values 0x00000001 and 0x00000002 can also be used for the RGB565 color mode configuration

**DSI\_RGB666**

The value 0x00000004 can also be used for the RGB666 color mode configuration

**DSI\_RGB888**

*DSI Color Mode*

**DSI\_COLOR\_MODE\_FULL**

**DSI\_COLOR\_MODE\_EIGHT**

*DSI Communication Delay*

**DSI\_SLEW\_RATE\_HSTX**

**DSI\_SLEW\_RATE\_LPTX**

**DSI\_HS\_DELAY**

*DSI CustomLane*

**DSI\_SWAP\_LANE\_PINS**

**DSI\_INVERT\_HS\_SIGNAL**

***DSI DATA ENABLE Polarity***

DSI\_DATA\_ENABLE\_ACTIVE\_HIGH

DSI\_DATA\_ENABLE\_ACTIVE\_LOW

***DSI DCS Command***

DSI\_ENTER\_IDLE\_MODE

DSI\_ENTER\_INVERT\_MODE

DSI\_ENTER\_NORMAL\_MODE

DSI\_ENTER\_PARTIAL\_MODE

DSI\_ENTER\_SLEEP\_MODE

DSI\_EXIT\_IDLE\_MODE

DSI\_EXIT\_INVERT\_MODE

DSI\_EXIT\_SLEEP\_MODE

DSI\_GET\_3D\_CONTROL

DSI\_GET\_ADDRESS\_MODE

DSI\_GET\_BLUE\_CHANNEL

DSI\_GET\_DIAGNOSTIC\_RESULT

DSI\_GET\_DISPLAY\_MODE

DSI\_GET\_GREEN\_CHANNEL

DSI\_GET\_PIXEL\_FORMAT

DSI\_GET\_POWER\_MODE

DSI\_GET\_RED\_CHANNEL

DSI\_GET\_SCANLINE

DSI\_GET\_SIGNAL\_MODE

DSI\_NOP

DSI\_READ\_DDB\_CONTINUE

DSI\_READ\_DDB\_START

DSI\_READ\_MEMORY\_CONTINUE

DSI\_READ\_MEMORY\_START

DSI\_SET\_3D\_CONTROL

DSI\_SET\_ADDRESS\_MODE

DSI\_SET\_COLUMN\_ADDRESS

DSI\_SET\_DISPLAY\_OFF

DSI\_SET\_DISPLAY\_ON

DSI\_SET\_GAMMA\_CURVE

DSI\_SET\_PAGE\_ADDRESS

DSI\_SET\_PARTIAL\_COLUMNS

DSI\_SET\_PARTIAL\_ROWS

DSI\_SET\_PIXEL\_FORMAT

DSI\_SET\_SCROLL\_AREA

DSI\_SET\_SCROLL\_START

DSI\_SET\_TEAR\_OFF

DSI\_SET\_TEAR\_ON

DSI\_SET\_TEAR\_SCANLINE

DSI\_SET\_VSYNC\_TIMING

DSI\_SOFT\_RESET

DSI\_WRITE\_LUT

DSI\_WRITE\_MEMORY\_CONTINUE

DSI\_WRITE\_MEMORY\_START

***DSI Error Data Type***

HAL\_DSI\_ERROR\_NONE

HAL\_DSI\_ERROR\_ACK

acknowledge errors

HAL\_DSI\_ERROR\_PHY

PHY related errors

HAL\_DSI\_ERROR\_TX

transmission error

HAL\_DSI\_ERROR\_RX

reception error

HAL\_DSI\_ERROR\_ECC

ECC errors

#### HAL\_DSI\_ERROR\_CRC

CRC error

#### HAL\_DSI\_ERROR\_PSE

Packet Size error

#### HAL\_DSI\_ERROR\_EOT

End Of Transmission error

#### HAL\_DSI\_ERROR\_OVF

FIFO overflow error

#### HAL\_DSI\_ERROR\_GEN

Generic FIFO related errors

#### **DSI Exported Macros**

#### \_\_HAL\_DSI\_RESET\_HANDLE\_STATE

##### **Description:**

- Reset DSI handle state.

##### **Parameters:**

- \_\_HANDLE\_\_: DSI handle

##### **Return value:**

- None

#### \_\_HAL\_DSI\_ENABLE

##### **Description:**

- Enables the DSI host.

##### **Parameters:**

- \_\_HANDLE\_\_: DSI handle

##### **Return value:**

- None.

#### \_\_HAL\_DSI\_DISABLE

##### **Description:**

- Disables the DSI host.

##### **Parameters:**

- \_\_HANDLE\_\_: DSI handle

##### **Return value:**

- None.

#### \_\_HAL\_DSI\_WRAPPER\_ENABLE

##### **Description:**

- Enables the DSI wrapper.

##### **Parameters:**

- \_\_HANDLE\_\_: DSI handle

##### **Return value:**

- None.

### **\_\_HAL\_DSI\_WRAPPER\_DISABLE**

**Description:**

- Disable the DSI wrapper.

**Parameters:**

- `__HANDLE__`: DSI handle

**Return value:**

- None.

### **\_\_HAL\_DSI\_PLL\_ENABLE**

**Description:**

- Enables the DSI PLL.

**Parameters:**

- `__HANDLE__`: DSI handle

**Return value:**

- None.

### **\_\_HAL\_DSI\_PLL\_DISABLE**

**Description:**

- Disables the DSI PLL.

**Parameters:**

- `__HANDLE__`: DSI handle

**Return value:**

- None.

### **\_\_HAL\_DSI\_REG\_ENABLE**

**Description:**

- Enables the DSI regulator.

**Parameters:**

- `__HANDLE__`: DSI handle

**Return value:**

- None.

### **\_\_HAL\_DSI\_REG\_DISABLE**

**Description:**

- Disables the DSI regulator.

**Parameters:**

- `__HANDLE__`: DSI handle

**Return value:**

- None.



### \_\_HAL\_DSI\_GET\_FLAG

**Description:**

- Get the DSI pending flags.

**Parameters:**

- `__HANDLE__`: DSI handle.
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
  - `DSI_FLAG_TE` : Tearing Effect Interrupt Flag
  - `DSI_FLAG_ER` : End of Refresh Interrupt Flag
  - `DSI_FLAG_BUSY` : Busy Flag
  - `DSI_FLAG_PLLLS`: PLL Lock Status
  - `DSI_FLAG_PLLL` : PLL Lock Interrupt Flag
  - `DSI_FLAG_PLLU` : PLL Unlock Interrupt Flag
  - `DSI_FLAG_RRS` : Regulator Ready Flag
  - `DSI_FLAG_RR` : Regulator Ready Interrupt Flag

**Return value:**

- The: state of FLAG (SET or RESET).

### \_\_HAL\_DSI\_CLEAR\_FLAG

**Description:**

- Clears the DSI pending flags.

**Parameters:**

- `__HANDLE__`: DSI handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DSI_FLAG_TE` : Tearing Effect Interrupt Flag
  - `DSI_FLAG_ER` : End of Refresh Interrupt Flag
  - `DSI_FLAG_PLLL` : PLL Lock Interrupt Flag
  - `DSI_FLAG_PLLU` : PLL Unlock Interrupt Flag
  - `DSI_FLAG_RR` : Regulator Ready Interrupt Flag

**Return value:**

- None

### \_\_HAL\_DSI\_ENABLE\_IT

**Description:**

- Enables the specified DSI interrupts.

**Parameters:**

- `__HANDLE__`: DSI handle.
- `__INTERRUPT__`: specifies the DSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - `DSI_IT_TE` : Tearing Effect Interrupt
  - `DSI_IT_ER` : End of Refresh Interrupt
  - `DSI_IT_PLLL`: PLL Lock Interrupt
  - `DSI_IT_PLLU`: PLL Unlock Interrupt
  - `DSI_IT_RR` : Regulator Ready Interrupt

**Return value:**

- None

## \_\_HAL\_DSI\_DISABLE\_IT

### Description:

- Disables the specified DSI interrupts.

### Parameters:

- \_\_HANDLE\_\_: DSI handle
- \_\_INTERRUPT\_\_: specifies the DSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DSI\_IT\_TE : Tearing Effect Interrupt
  - DSI\_IT\_ER : End of Refresh Interrupt
  - DSI\_IT\_PLLL: PLL Lock Interrupt
  - DSI\_IT\_PLLU: PLL Unlock Interrupt
  - DSI\_IT\_RR : Regulator Ready Interrupt

### Return value:

- None

## \_\_HAL\_DSI\_GET\_IT\_SOURCE

### Description:

- Checks whether the specified DSI interrupt source is enabled or not.

### Parameters:

- \_\_HANDLE\_\_: DSI handle
- \_\_INTERRUPT\_\_: specifies the DSI interrupt source to check. This parameter can be one of the following values:
  - DSI\_IT\_TE : Tearing Effect Interrupt
  - DSI\_IT\_ER : End of Refresh Interrupt
  - DSI\_IT\_PLLL: PLL Lock Interrupt
  - DSI\_IT\_PLLU: PLL Unlock Interrupt
  - DSI\_IT\_RR : Regulator Ready Interrupt

### Return value:

- The: state of INTERRUPT (SET or RESET).

### *DSI FBTA Acknowledge*

## DSI\_FBTAA\_DISABLE

## DSI\_FBTAA\_ENABLE

### *DSI Flags*

## DSI\_FLAG\_TE

## DSI\_FLAG\_ER

## DSI\_FLAG\_BUSY

## DSI\_FLAG\_PLLLS

## DSI\_FLAG\_PLLL

## DSI\_FLAG\_PLLU

## DSI\_FLAG\_RRS

## DSI\_FLAG\_RR

### *DSI Flow Control*

DSI\_FLOW\_CONTROL\_CRC\_RX

DSI\_FLOW\_CONTROL\_ECC\_RX

DSI\_FLOW\_CONTROL\_BTA

DSI\_FLOW\_CONTROL\_EOTP\_RX

DSI\_FLOW\_CONTROL\_EOTP\_TX

DSI\_FLOW\_CONTROL\_ALL

*DSI HSYNC Polarity*

DSI\_HSYNC\_ACTIVE\_HIGH

DSI\_HSYNC\_ACTIVE\_LOW

*DSI HS Presp Mode*

DSI\_HS\_PM\_DISABLE

DSI\_HS\_PM\_ENABLE

*DSI Interrupts*

DSI\_IT\_TE

DSI\_IT\_ER

DSI\_IT\_PLLL

DSI\_IT\_PLLU

DSI\_IT\_RR

*DSI Lane Group*

DSI\_CLOCK\_LANE

DSI\_DATA\_LANES

*DSI Lane Select*

DSI\_CLK\_LANE

DSI\_DATA\_LANE0

DSI\_DATA\_LANE1

*DSI LONG WRITE PKT Data Type*

DSI\_DCS\_LONG\_PKT\_WRITE

DCS long write

DSI\_GEN\_LONG\_PKT\_WRITE

Generic long write

*DSI Loosely Packed*

DSI\_LOOSELY\_PACKED\_ENABLE

DSI\_LOOSELY\_PACKED\_DISABLE

*DSI LP Command*

DSI\_LP\_COMMAND\_DISABLE

DSI\_LP\_COMMAND\_ENABLE

*DSI LP HBP*

DSI\_LP\_HBP\_DISABLE

DSI\_LP\_HBP\_ENABLE

*DSI LP HFP*

DSI\_LP\_HFP\_DISABLE

DSI\_LP\_HFP\_ENABLE

*DSI LP LPDcs Long Write*

DSI\_LP\_DLW\_DISABLE

DSI\_LP\_DLW\_ENABLE

*DSI LP LPDcs Short Read NoP*

DSI\_LP\_DSR0P\_DISABLE

DSI\_LP\_DSR0P\_ENABLE

*DSI LP LPDcs Short Write NoP*

DSI\_LP\_DSW0P\_DISABLE

DSI\_LP\_DSW0P\_ENABLE

*DSI LP LPDcs Short Write OneP*

DSI\_LP\_DSW1P\_DISABLE

DSI\_LP\_DSW1P\_ENABLE

*DSI LP LPGen LongWrite*

DSI\_LP\_GLW\_DISABLE

DSI\_LP\_GLW\_ENABLE

*DSI LP LPGen Short Read NoP*

DSI\_LP\_GSR0P\_DISABLE

DSI\_LP\_GSR0P\_ENABLE

*DSI LP LPGen Short Read OneP*

DSI\_LP\_GSR1P\_DISABLE

DSI\_LP\_GSR1P\_ENABLE

*DSI LP LPGen Short Read TwoP*

DSI\_LP\_GSR2P\_DISABLE

DSI\_LP\_GSR2P\_ENABLE

*DSI LP LPGen Short Write NoP*

DSI\_LP\_GSW0P\_DISABLE

DSI\_LP\_GSW0P\_ENABLE

*DSI LP LPGen Short Write OneP*

DSI\_LP\_GSW1P\_DISABLE

DSI\_LP\_GSW1P\_ENABLE

*DSI LP LPGen Short Write TwoP*

DSI\_LP\_GSW2P\_DISABLE

DSI\_LP\_GSW2P\_ENABLE

*DSI LP LPMax Read Packet*

DSI\_LP\_MRDP\_DISABLE

DSI\_LP\_MRDP\_ENABLE

*DSI LP VACT*

DSI\_LP\_VACT\_DISABLE

DSI\_LP\_VACT\_ENABLE

*DSI LP VBP*

DSI\_LP\_VBP\_DISABLE

DSI\_LP\_VBP\_ENABLE

*DSI LP VFP*

DSI\_LP\_VFP\_DISABLE

DSI\_LP\_VFP\_ENABLE

*DSI LP VSYNC*

DSI\_LP\_VSYNC\_DISABLE

DSI\_LP\_VSYNC\_ENABLE

*DSI Number Of Lanes*

DSI\_ONE\_DATA\_LANE

DSI\_TWO\_DATA\_LANES

*DSI PHY Timing*

DSI\_TCLK\_POST

DSI\_TLPX\_CLK

DSI\_THS\_EXIT

DSI\_TLPX\_DATA

DSI\_THS\_ZERO

DSI\_THS\_TRAIL

DSI\_THS\_PREPARE

DSI\_TCLK\_ZERO

DSI\_TCLK\_PREPARE

***DSI PLL IDF***

DSI\_PLL\_IN\_DIV1

DSI\_PLL\_IN\_DIV2

DSI\_PLL\_IN\_DIV3

DSI\_PLL\_IN\_DIV4

DSI\_PLL\_IN\_DIV5

DSI\_PLL\_IN\_DIV6

DSI\_PLL\_IN\_DIV7

***DSI PLL ODF***

DSI\_PLL\_OUT\_DIV1

DSI\_PLL\_OUT\_DIV2

DSI\_PLL\_OUT\_DIV4

DSI\_PLL\_OUT\_DIV8

***DSI SHORT READ PKT Data Type***

DSI\_DCS\_SHORT\_PKT\_READ

DCS short read

DSI\_GEN\_SHORT\_PKT\_READ\_P0

Generic short read, no parameters

DSI\_GEN\_SHORT\_PKT\_READ\_P1

Generic short read, one parameter

DSI\_GEN\_SHORT\_PKT\_READ\_P2

Generic short read, two parameters

***DSI SHORT WRITE PKT Data Type***

DSI\_DCS\_SHORT\_PKT\_WRITE\_P0

DCS short write, no parameters

DSI\_DCS\_SHORT\_PKT\_WRITE\_P1

DCS short write, one parameter

DSI\_GEN\_SHORT\_PKT\_WRITE\_P0

Generic short write, no parameters

DSI\_GEN\_SHORT\_PKT\_WRITE\_P1

Generic short write, one parameter

DSI\_GEN\_SHORT\_PKT\_WRITE\_P2

Generic short write, two parameters

***DSI ShutDown***

DSI\_DISPLAY\_ON

DSI\_DISPLAY\_OFF

***DSI Tearing Effect Polarity***

DSI\_TE\_RISING\_EDGE

DSI\_TE\_FALLING\_EDGE

***DSI Tearing Effect Source***

DSI\_TE\_DSILINK

DSI\_TE\_EXTERNAL

***DSI TE Acknowledge Request***

DSI\_TE\_ACKNOWLEDGE\_DISABLE

DSI\_TE\_ACKNOWLEDGE\_ENABLE

***DSI Video Mode Type***

DSI\_VID\_MODE\_NB\_PULSES

DSI\_VID\_MODE\_NB\_EVENTS

DSI\_VID\_MODE\_BURST

***DSI VSYNC Active Polarity***

DSI\_VSYNC\_ACTIVE\_HIGH

DSI\_VSYNC\_ACTIVE\_LOW

***DSI Vsync Polarity***

DSI\_VSYNC\_FALLING

DSI\_VSYNC\_RISING

## 24 HAL ETH Generic Driver

### 24.1 ETH Firmware driver registers structures

#### 24.1.1 ETH\_InitTypeDef

*ETH\_InitTypeDef* is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- *uint32\_t* *AutoNegotiation*
- *uint32\_t* *Speed*
- *uint32\_t* *DuplexMode*
- *uint16\_t* *PhyAddress*
- *uint8\_t* \* *MACAddr*
- *uint32\_t* *RxMode*
- *uint32\_t* *ChecksumMode*
- *uint32\_t* *MedialInterface*

##### Field Documentation

- *uint32\_t* *ETH\_InitTypeDef::AutoNegotiation*  
 Selects or not the AutoNegotiation mode for the external PHY The AutoNegotiation allows an automatic setting of the Speed (10/100Mbps) and the mode (half/full-duplex). This parameter can be a value of [ETH\\_AutoNegotiation](#)
- *uint32\_t* *ETH\_InitTypeDef::Speed*  
 Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of [ETH\\_Speed](#)
- *uint32\_t* *ETH\_InitTypeDef::DuplexMode*  
 Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of [ETH\\_Duplex\\_Mode](#)
- *uint16\_t* *ETH\_InitTypeDef::PhyAddress*  
 Ethernet PHY address. This parameter must be a number between `Min_Data = 0` and `Max_Data = 32`
- *uint8\_t* \* *ETH\_InitTypeDef::MACAddr*  
 MAC Address of used Hardware: must be pointer on an array of 6 bytes
- *uint32\_t* *ETH\_InitTypeDef::RxMode*  
 Selects the Ethernet Rx mode: Polling mode, Interrupt mode. This parameter can be a value of [ETH\\_Rx\\_Mode](#)
- *uint32\_t* *ETH\_InitTypeDef::ChecksumMode*  
 Selects if the checksum is check by hardware or by software. This parameter can be a value of [ETH\\_Checksum\\_Mode](#)
- *uint32\_t* *ETH\_InitTypeDef::MedialInterface*  
 Selects the media-independent interface or the reduced media-independent interface. This parameter can be a value of [ETH\\_Media\\_Interface](#)

#### 24.1.2 ETH\_MACInitTypeDef

*ETH\_MACInitTypeDef* is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- *uint32\_t* *Watchdog*
- *uint32\_t* *Jabber*
- *uint32\_t* *InterFrameGap*
- *uint32\_t* *CarrierSense*
- *uint32\_t* *ReceiveOwn*
- *uint32\_t* *LoopbackMode*
- *uint32\_t* *ChecksumOffload*
- *uint32\_t* *RetryTransmission*



- *uint32\_t AutomaticPadCRCStrip*
- *uint32\_t BackOffLimit*
- *uint32\_t DeferralCheck*
- *uint32\_t ReceiveAll*
- *uint32\_t SourceAddrFilter*
- *uint32\_t PassControlFrames*
- *uint32\_t BroadcastFramesReception*
- *uint32\_t DestinationAddrFilter*
- *uint32\_t PromiscuousMode*
- *uint32\_t MulticastFramesFilter*
- *uint32\_t UnicastFramesFilter*
- *uint32\_t HashTableHigh*
- *uint32\_t HashTableLow*
- *uint32\_t PauseTime*
- *uint32\_t ZeroQuantaPause*
- *uint32\_t PauseLowThreshold*
- *uint32\_t UnicastPauseFrameDetect*
- *uint32\_t ReceiveFlowControl*
- *uint32\_t TransmitFlowControl*
- *uint32\_t VLANTagComparison*
- *uint32\_t VLANTagIdentifier*

#### Field Documentation

- *uint32\_t ETH\_MACInitTypeDef::Watchdog*  
 Selects or not the Watchdog timer. When enabled, the MAC allows no more than 2048 bytes to be received. When disabled, the MAC can receive up to 16384 bytes. This parameter can be a value of [ETH\\_Watchdog](#)
- *uint32\_t ETH\_MACInitTypeDef::Jabber*  
 Selects or not Jabber timer. When enabled, the MAC allows no more than 2048 bytes to be sent. When disabled, the MAC can send up to 16384 bytes. This parameter can be a value of [ETH\\_Jabber](#)
- *uint32\_t ETH\_MACInitTypeDef::InterFrameGap*  
 Selects the minimum IFG between frames during transmission. This parameter can be a value of [ETH\\_Inter\\_Frame\\_Gap](#)
- *uint32\_t ETH\_MACInitTypeDef::CarrierSense*  
 Selects or not the Carrier Sense. This parameter can be a value of [ETH\\_Carrier\\_Sense](#)
- *uint32\_t ETH\_MACInitTypeDef::ReceiveOwn*  
 Selects or not the ReceiveOwn, ReceiveOwn allows the reception of frames when the TX\_EN signal is asserted in Half-Duplex mode. This parameter can be a value of [ETH\\_Receive\\_Own](#)
- *uint32\_t ETH\_MACInitTypeDef::LoopbackMode*  
 Selects or not the internal MAC MII Loopback mode. This parameter can be a value of [ETH\\_Loop\\_Back\\_Mode](#)
- *uint32\_t ETH\_MACInitTypeDef::ChecksumOffload*  
 Selects or not the IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. This parameter can be a value of [ETH\\_Checksum\\_Offload](#)
- *uint32\_t ETH\_MACInitTypeDef::RetryTransmission*  
 Selects or not the MAC attempt retries transmission, based on the settings of BL, when a collision occurs (Half-Duplex mode). This parameter can be a value of [ETH\\_Retry\\_Transmission](#)
- *uint32\_t ETH\_MACInitTypeDef::AutomaticPadCRCStrip*  
 Selects or not the Automatic MAC Pad/CRC Stripping. This parameter can be a value of [ETH\\_Automatic\\_Pad\\_CRC\\_Strip](#)
- *uint32\_t ETH\_MACInitTypeDef::BackOffLimit*  
 Selects the BackOff limit value. This parameter can be a value of [ETH\\_Back\\_Off\\_Limit](#)

- ***uint32\_t ETH\_MACInitTypeDef::DeferralCheck***  
 Selects or not the deferral check function (Half-Duplex mode). This parameter can be a value of [ETH\\_Deferral\\_Check](#)
- ***uint32\_t ETH\_MACInitTypeDef::ReceiveAll***  
 Selects or not all frames reception by the MAC (No filtering). This parameter can be a value of [ETH\\_Receive\\_All](#)
- ***uint32\_t ETH\_MACInitTypeDef::SourceAddrFilter***  
 Selects the Source Address Filter mode. This parameter can be a value of [ETH\\_Source\\_Addr\\_Filter](#)
- ***uint32\_t ETH\_MACInitTypeDef::PassControlFrames***  
 Sets the forwarding mode of the control frames (including unicast and multicast PAUSE frames) This parameter can be a value of [ETH\\_Pass\\_Control\\_Frames](#)
- ***uint32\_t ETH\_MACInitTypeDef::BroadcastFramesReception***  
 Selects or not the reception of Broadcast Frames. This parameter can be a value of [ETH\\_Broadcast\\_Frames\\_Reception](#)
- ***uint32\_t ETH\_MACInitTypeDef::DestinationAddrFilter***  
 Sets the destination filter mode for both unicast and multicast frames. This parameter can be a value of [ETH\\_Destination\\_Addr\\_Filter](#)
- ***uint32\_t ETH\_MACInitTypeDef::PromiscuousMode***  
 Selects or not the Promiscuous Mode This parameter can be a value of [ETH\\_Promiscuous\\_Mode](#)
- ***uint32\_t ETH\_MACInitTypeDef::MulticastFramesFilter***  
 Selects the Multicast Frames filter mode: None/HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of [ETH\\_Multicast\\_Frames\\_Filter](#)
- ***uint32\_t ETH\_MACInitTypeDef::UnicastFramesFilter***  
 Selects the Unicast Frames filter mode: HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of [ETH\\_Unicast\\_Frames\\_Filter](#)
- ***uint32\_t ETH\_MACInitTypeDef::HashTableHigh***  
 This field holds the higher 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFFU
- ***uint32\_t ETH\_MACInitTypeDef::HashTableLow***  
 This field holds the lower 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFFU
- ***uint32\_t ETH\_MACInitTypeDef::PauseTime***  
 This field holds the value to be used in the Pause Time field in the transmit control frame. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFU
- ***uint32\_t ETH\_MACInitTypeDef::ZeroQuantaPause***  
 Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of [ETH\\_Zero\\_Quanta\\_Pause](#)
- ***uint32\_t ETH\_MACInitTypeDef::PauseLowThreshold***  
 This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of [ETH\\_Pause\\_Low\\_Threshold](#)
- ***uint32\_t ETH\_MACInitTypeDef::UnicastPauseFrameDetect***  
 Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast address and unique multicast address). This parameter can be a value of [ETH\\_Unicast\\_Pause\\_Frame\\_Detect](#)
- ***uint32\_t ETH\_MACInitTypeDef::ReceiveFlowControl***  
 Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of [ETH\\_Receive\\_Flow\\_Control](#)
- ***uint32\_t ETH\_MACInitTypeDef::TransmitFlowControl***  
 Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of [ETH\\_Transmit\\_Flow\\_Control](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagComparison***  
 Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of [ETH\\_VLAN\\_Tag\\_Comparison](#)

- **`uint32_t ETH_MACInitTypeDef::VLANTagIdentifier`**  
Holds the VLAN tag identifier for receive frames

### 24.1.3

#### ETH\_DMAInitTypeDef

*ETH\_DMAInitTypeDef* is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- **`uint32_t DropTCPIPChecksumErrorFrame`**
- **`uint32_t ReceiveStoreForward`**
- **`uint32_t FlushReceivedFrame`**
- **`uint32_t TransmitStoreForward`**
- **`uint32_t TransmitThresholdControl`**
- **`uint32_t ForwardErrorFrames`**
- **`uint32_t ForwardUndersizedGoodFrames`**
- **`uint32_t ReceiveThresholdControl`**
- **`uint32_t SecondFrameOperate`**
- **`uint32_t AddressAlignedBeats`**
- **`uint32_t FixedBurst`**
- **`uint32_t RxDMA BurstLength`**
- **`uint32_t TxDMA BurstLength`**
- **`uint32_t EnhancedDescriptorFormat`**
- **`uint32_t DescriptorSkipLength`**
- **`uint32_t DMA Arbitration`**

##### Field Documentation

- **`uint32_t ETH_DMAInitTypeDef::DropTCPIPChecksumErrorFrame`**  
Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of [ETH\\_Drop\\_TCP\\_IP\\_Checksum\\_Error\\_Frame](#)
- **`uint32_t ETH_DMAInitTypeDef::ReceiveStoreForward`**  
Enables or disables the Receive store and forward mode. This parameter can be a value of [ETH\\_Receive\\_Store\\_Forward](#)
- **`uint32_t ETH_DMAInitTypeDef::FlushReceivedFrame`**  
Enables or disables the flushing of received frames. This parameter can be a value of [ETH\\_Flush\\_Received\\_Frame](#)
- **`uint32_t ETH_DMAInitTypeDef::TransmitStoreForward`**  
Enables or disables Transmit store and forward mode. This parameter can be a value of [ETH\\_Transmit\\_Store\\_Forward](#)
- **`uint32_t ETH_DMAInitTypeDef::TransmitThresholdControl`**  
Selects or not the Transmit Threshold Control. This parameter can be a value of [ETH\\_Transmit\\_Threshold\\_Control](#)
- **`uint32_t ETH_DMAInitTypeDef::ForwardErrorFrames`**  
Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of [ETH\\_Forward\\_Error\\_Frames](#)
- **`uint32_t ETH_DMAInitTypeDef::ForwardUndersizedGoodFrames`**  
Enables or disables the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC) This parameter can be a value of [ETH\\_Forward\\_Undersized\\_Good\\_Frames](#)
- **`uint32_t ETH_DMAInitTypeDef::ReceiveThresholdControl`**  
Selects the threshold level of the Receive FIFO. This parameter can be a value of [ETH\\_Receive\\_Threshold\\_Control](#)
- **`uint32_t ETH_DMAInitTypeDef::SecondFrameOperate`**  
Selects or not the Operate on second frame mode, which allows the DMA to process a second frame of Transmit data even before obtaining the status for the first frame. This parameter can be a value of [ETH\\_Second\\_Frame\\_Operate](#)

- ***uint32\_t ETH\_DMAInitTypeDef::AddressAlignedBeats***  
Enables or disables the Address Aligned Beats. This parameter can be a value of [ETH\\_Address\\_Aligned\\_Beats](#)
- ***uint32\_t ETH\_DMAInitTypeDef::FixedBurst***  
Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of [ETH\\_Fixed\\_Burst](#)
- ***uint32\_t ETH\_DMAInitTypeDef::RxDMABurstLength***  
Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of [ETH\\_Rx\\_DMA\\_Burst\\_Length](#)
- ***uint32\_t ETH\_DMAInitTypeDef::TxDMABurstLength***  
Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of [ETH\\_Tx\\_DMA\\_Burst\\_Length](#)
- ***uint32\_t ETH\_DMAInitTypeDef::EnhancedDescriptorFormat***  
Enables the enhanced descriptor format. This parameter can be a value of [ETH\\_DMA\\_Enhanced\\_descriptor\\_format](#)
- ***uint32\_t ETH\_DMAInitTypeDef::DescriptorSkipLength***  
Specifies the number of word to skip between two unchained descriptors (Ring mode) This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- ***uint32\_t ETH\_DMAInitTypeDef::DMAArbitration***  
Selects the DMA Tx/Rx arbitration. This parameter can be a value of [ETH\\_DMA\\_Arbitration](#)

#### 24.1.4

#### **ETH\_DMADescTypeDef**

**ETH\_DMADescTypeDef** is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- ***\_\_IO uint32\_t Status***
- ***uint32\_t ControlBufferSize***
- ***uint32\_t Buffer1Addr***
- ***uint32\_t Buffer2NextDescAddr***
- ***uint32\_t ExtendedStatus***
- ***uint32\_t Reserved1***
- ***uint32\_t TimeStampLow***
- ***uint32\_t TimeStampHigh***

##### Field Documentation

- ***\_\_IO uint32\_t ETH\_DMADescTypeDef::Status***  
Status
- ***uint32\_t ETH\_DMADescTypeDef::ControlBufferSize***  
Control and Buffer1, Buffer2 lengths
- ***uint32\_t ETH\_DMADescTypeDef::Buffer1Addr***  
Buffer1 address pointer
- ***uint32\_t ETH\_DMADescTypeDef::Buffer2NextDescAddr***  
Buffer2 or next descriptor address pointer Enhanced ETHERNET DMA PTP Descriptors
- ***uint32\_t ETH\_DMADescTypeDef::ExtendedStatus***  
Extended status for PTP receive descriptor
- ***uint32\_t ETH\_DMADescTypeDef::Reserved1***  
Reserved
- ***uint32\_t ETH\_DMADescTypeDef::TimeStampLow***  
Time Stamp Low value for transmit and receive
- ***uint32\_t ETH\_DMADescTypeDef::TimeStampHigh***  
Time Stamp High value for transmit and receive

### 24.1.5 ETH\_DMARxFramelInfos

*ETH\_DMARxFramelInfos* is defined in the stm32f4xx\_hal\_eth.h

#### Data Fields

- *ETH\_DMADescTypeDef* \* *FSRxDesc*
- *ETH\_DMADescTypeDef* \* *LSRxDesc*
- *uint32\_t* *SegCount*
- *uint32\_t* *length*
- *uint32\_t* *buffer*

#### Field Documentation

- *ETH\_DMADescTypeDef*\* *ETH\_DMARxFramelInfos::FSRxDesc*  
First Segment Rx Desc
- *ETH\_DMADescTypeDef*\* *ETH\_DMARxFramelInfos::LSRxDesc*  
Last Segment Rx Desc
- *uint32\_t* *ETH\_DMARxFramelInfos::SegCount*  
Segment count
- *uint32\_t* *ETH\_DMARxFramelInfos::length*  
Frame length
- *uint32\_t* *ETH\_DMARxFramelInfos::buffer*  
Frame buffer

### 24.1.6 ETH\_HandleTypeDef

*ETH\_HandleTypeDef* is defined in the stm32f4xx\_hal\_eth.h

#### Data Fields

- *ETH\_TypeDef* \* *Instance*
- *ETH\_InitTypeDef* *Init*
- *uint32\_t* *LinkStatus*
- *ETH\_DMADescTypeDef* \* *RxDesc*
- *ETH\_DMADescTypeDef* \* *TxDesc*
- *ETH\_DMARxFramelInfos* *RxFramelInfos*
- *\_\_IO HAL\_ETH\_StateTypeDef* *State*
- *HAL\_LockTypeDef* *Lock*

#### Field Documentation

- *ETH\_TypeDef*\* *ETH\_HandleTypeDef::Instance*  
Register base address
- *ETH\_InitTypeDef* *ETH\_HandleTypeDef::Init*  
Ethernet Init Configuration
- *uint32\_t* *ETH\_HandleTypeDef::LinkStatus*  
Ethernet link status
- *ETH\_DMADescTypeDef*\* *ETH\_HandleTypeDef::RxDesc*  
Rx descriptor to Get
- *ETH\_DMADescTypeDef*\* *ETH\_HandleTypeDef::TxDesc*  
Tx descriptor to Set
- *ETH\_DMARxFramelInfos* *ETH\_HandleTypeDef::RxFramelInfos*  
last Rx frame infos
- *\_\_IO HAL\_ETH\_StateTypeDef* *ETH\_HandleTypeDef::State*  
ETH communication state
- *HAL\_LockTypeDef* *ETH\_HandleTypeDef::Lock*  
ETH Lock

## 24.2 ETH Firmware driver API description

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

### 24.2.1 How to use this driver

1. Declare a `ETH_HandleTypeDef` handle structure, for example: `ETH_HandleTypeDef heth;`
2. Fill parameters of `Init` structure in `heth` handle
3. Call `HAL_ETH_Init()` API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the `HAL_ETH_MspInit()` API:
  - a. Enable the Ethernet interface clock using
    - `__HAL_RCC_ETHMAC_CLK_ENABLE();`
    - `__HAL_RCC_ETHMACTX_CLK_ENABLE();`
    - `__HAL_RCC_ETHMACRX_CLK_ENABLE();`
  - b. Initialize the related GPIO clocks
  - c. Configure Ethernet pin-out
  - d. Configure Ethernet NVIC interrupt (IT mode)
5. Initialize Ethernet DMA Descriptors in chain mode and point to allocated buffers:
  - a. `HAL_ETH_DMATxDescListInit();` for Transmission process
  - b. `HAL_ETH_DMARxDescListInit();` for Reception process
6. Enable MAC and DMA transmission and reception:
  - a. `HAL_ETH_Start();`
7. Prepare ETH DMA TX Descriptors and give the hand to ETH DMA to transfer the frame to MAC TX FIFO:
  - a. `HAL_ETH_TransmitFrame();`
8. Poll for a received frame in ETH RX DMA Descriptors and get received frame parameters
  - a. `HAL_ETH_GetReceivedFrame();` (should be called into an infinite loop)
9. Get a received frame when an ETH RX interrupt occurs:
  - a. `HAL_ETH_GetReceivedFrame_IT();` (called in IT mode only)
10. Communicate with external PHY device:
  - a. Read a specific register from the PHY `HAL_ETH_ReadPHYRegister();`
  - b. Write data to a specific RHY register: `HAL_ETH_WritePHYRegister();`
11. Configure the Ethernet MAC after ETH peripheral initialization `HAL_ETH_ConfigMAC();` all MAC parameters should be filled.
12. Configure the Ethernet DMA after ETH peripheral initialization `HAL_ETH_ConfigDMA();` all DMA parameters should be filled.

*Note:* The PTP protocol and the DMA descriptors ring mode are not supported in this driver

### Callback registration

### 24.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the Ethernet peripheral
- De-initialize the Ethernet peripheral

This section contains the following APIs:

- [\*HAL\\_ETH\\_Init\(\)\*](#)
- [\*HAL\\_ETH\\_DeInit\(\)\*](#)
- [\*HAL\\_ETH\\_DMATxDescListInit\(\)\*](#)
- [\*HAL\\_ETH\\_DMARxDescListInit\(\)\*](#)
- [\*HAL\\_ETH\\_MspInit\(\)\*](#)
- [\*HAL\\_ETH\\_MspDeInit\(\)\*](#)

### 24.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a frame HAL\_ETH\_TransmitFrame();
- Receive a frame HAL\_ETH\_GetReceivedFrame(); HAL\_ETH\_GetReceivedFrame\_IT();
- Read from an External PHY register HAL\_ETH\_ReadPHYRegister();
- Write to an External PHY register HAL\_ETH\_WritePHYRegister();

This section contains the following APIs:

- *HAL\_ETH\_TransmitFrame()*
- *HAL\_ETH\_GetReceivedFrame()*
- *HAL\_ETH\_GetReceivedFrame\_IT()*
- *HAL\_ETH\_IRQHandler()*
- *HAL\_ETH\_TxCpltCallback()*
- *HAL\_ETH\_RxCpltCallback()*
- *HAL\_ETH\_ErrorCallback()*
- *HAL\_ETH\_ReadPHYRegister()*
- *HAL\_ETH\_WritePHYRegister()*

#### 24.2.4 Peripheral Control functions

This section provides functions allowing to:

- Enable MAC and DMA transmission and reception. HAL\_ETH\_Start();
- Disable MAC and DMA transmission and reception. HAL\_ETH\_Stop();
- Set the MAC configuration in runtime mode HAL\_ETH\_ConfigMAC();
- Set the DMA configuration in runtime mode HAL\_ETH\_ConfigDMA();

This section contains the following APIs:

- *HAL\_ETH\_Start()*
- *HAL\_ETH\_Stop()*
- *HAL\_ETH\_ConfigMAC()*
- *HAL\_ETH\_ConfigDMA()*

#### 24.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- Get the ETH handle state: HAL\_ETH\_GetState();

This section contains the following APIs:

- *HAL\_ETH\_GetState()*

#### 24.2.6 Detailed description of functions

##### HAL\_ETH\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_ETH\_Init (ETH\_HandleTypeDef \* heth)**

###### Function description

Initializes the Ethernet MAC and DMA according to default parameters.

###### Parameters

- **heth**: pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

###### Return values

- **HAL**: status

## HAL\_ETH\_DeInit

### Function name

**HAL\_StatusTypeDef HAL\_ETH\_DeInit (ETH\_HandleTypeDef \* heth)**

### Function description

De-Initializes the ETH peripheral.

### Parameters

- **heth**: pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

### Return values

- **HAL**: status

## HAL\_ETH\_MspInit

### Function name

**void HAL\_ETH\_MspInit (ETH\_HandleTypeDef \* heth)**

### Function description

Initializes the ETH MSP.

### Parameters

- **heth**: pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

### Return values

- **None**:

## HAL\_ETH\_MspDeInit

### Function name

**void HAL\_ETH\_MspDeInit (ETH\_HandleTypeDef \* heth)**

### Function description

Deinitializes ETH MSP.

### Parameters

- **heth**: pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

### Return values

- **None**:

## HAL\_ETH\_DMATxDescListInit

### Function name

**HAL\_StatusTypeDef HAL\_ETH\_DMATxDescListInit (ETH\_HandleTypeDef \* heth, ETH\_DMADescTypeDef \* DMATxDescTab, uint8\_t \* TxBuff, uint32\_t TxBuffCount)**

### Function description

Initializes the DMA Tx descriptors in chain mode.



### Parameters

- **heth**: pointer to a `ETH_HandleTypeDef` structure that contains the configuration information for ETHERNET module
- **DMATxDescTab**: Pointer to the first Tx desc list
- **TxBuff**: Pointer to the first TxBuffer list
- **TxBuffCount**: Number of the used Tx desc in the list

### Return values

- **HAL**: status

#### HAL\_ETH\_DMARxDescListInit

### Function name

`HAL_StatusTypeDef HAL_ETH_DMARxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMARxDescTab, uint8_t * RxBuff, uint32_t RxBuffCount)`

### Function description

Initializes the DMA Rx descriptors in chain mode.

### Parameters

- **heth**: pointer to a `ETH_HandleTypeDef` structure that contains the configuration information for ETHERNET module
- **DMARxDescTab**: Pointer to the first Rx desc list
- **RxBuff**: Pointer to the first RxBuffer list
- **RxBuffCount**: Number of the used Rx desc in the list

### Return values

- **HAL**: status

#### HAL\_ETH\_TransmitFrame

### Function name

`HAL_StatusTypeDef HAL_ETH_TransmitFrame (ETH_HandleTypeDef * heth, uint32_t FrameLength)`

### Function description

Sends an Ethernet frame.

### Parameters

- **heth**: pointer to a `ETH_HandleTypeDef` structure that contains the configuration information for ETHERNET module
- **FrameLength**: Amount of data to be sent

### Return values

- **HAL**: status

#### HAL\_ETH\_GetReceivedFrame

### Function name

`HAL_StatusTypeDef HAL_ETH_GetReceivedFrame (ETH_HandleTypeDef * heth)`

### Function description

Checks for received frames.

### Parameters

- **heth**: pointer to a `ETH_HandleTypeDef` structure that contains the configuration information for ETHERNET module

### Return values

- **HAL:** status

### HAL\_ETH\_ReadPHYRegister

### Function name

**HAL\_StatusTypeDef HAL\_ETH\_ReadPHYRegister (ETH\_HandleTypeDef \* heth, uint16\_t PHYReg, uint32\_t \* RegValue)**

### Function description

Reads a PHY register.

### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYReg:** PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY\_BCR: Transceiver Basic Control Register, PHY\_BSR: Transceiver Basic Status Register. More PHY register could be read depending on the used PHY
- **RegValue:** PHY register value

### Return values

- **HAL:** status

### HAL\_ETH\_WritePHYRegister

### Function name

**HAL\_StatusTypeDef HAL\_ETH\_WritePHYRegister (ETH\_HandleTypeDef \* heth, uint16\_t PHYReg, uint32\_t RegValue)**

### Function description

Writes to a PHY register.

### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYReg:** PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY\_BCR: Transceiver Control Register. More PHY register could be written depending on the used PHY
- **RegValue:** the value to write

### Return values

- **HAL:** status

### HAL\_ETH\_GetReceivedFrame\_IT

### Function name

**HAL\_StatusTypeDef HAL\_ETH\_GetReceivedFrame\_IT (ETH\_HandleTypeDef \* heth)**

### Function description

Gets the Received frame in interrupt mode.

### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

### Return values

- **HAL:** status

### HAL\_ETH\_IRQHandler

#### Function name

**void HAL\_ETH\_IRQHandler (ETH\_HandleTypeDef \* heth)**

#### Function description

This function handles ETH interrupt request.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

#### Return values

- **HAL:** status

### HAL\_ETH\_TxCpltCallback

#### Function name

**void HAL\_ETH\_TxCpltCallback (ETH\_HandleTypeDef \* heth)**

#### Function description

Tx Transfer completed callbacks.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

#### Return values

- **None:**

### HAL\_ETH\_RxCpltCallback

#### Function name

**void HAL\_ETH\_RxCpltCallback (ETH\_HandleTypeDef \* heth)**

#### Function description

Rx Transfer completed callbacks.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

#### Return values

- **None:**

### HAL\_ETH\_ErrorCallback

#### Function name

**void HAL\_ETH\_ErrorCallback (ETH\_HandleTypeDef \* heth)**

#### Function description

Ethernet transfer error callbacks.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

#### Return values

- **None:**

**HAL\_ETH\_Start**

#### Function name

**HAL\_StatusTypeDef HAL\_ETH\_Start (ETH\_HandleTypeDef \* heth)**

#### Function description

Enables Ethernet MAC and DMA reception/transmission.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

#### Return values

- **HAL:** status

**HAL\_ETH\_Stop**

#### Function name

**HAL\_StatusTypeDef HAL\_ETH\_Stop (ETH\_HandleTypeDef \* heth)**

#### Function description

Stop Ethernet MAC and DMA reception/transmission.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

#### Return values

- **HAL:** status

**HAL\_ETH\_ConfigMAC**

#### Function name

**HAL\_StatusTypeDef HAL\_ETH\_ConfigMAC (ETH\_HandleTypeDef \* heth, ETH\_MACInitTypeDef \* macconf)**

#### Function description

Set ETH MAC Configuration.

#### Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** MAC Configuration structure

#### Return values

- **HAL:** status

**HAL\_ETH\_ConfigDMA**

#### Function name

**HAL\_StatusTypeDef HAL\_ETH\_ConfigDMA (ETH\_HandleTypeDef \* heth, ETH\_DMAInitTypeDef \* dmaconf)**

#### Function description

Sets ETH DMA Configuration.

#### Parameters

- **heth**: pointer to a `ETH_HandleTypeDef` structure that contains the configuration information for ETHERNET module
- **dmaconf**: DMA Configuration structure

#### Return values

- **HAL**: status

#### HAL\_ETH\_GetState

#### Function name

`HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)`

#### Function description

Return the ETH HAL state.

#### Parameters

- **heth**: pointer to a `ETH_HandleTypeDef` structure that contains the configuration information for ETHERNET module

#### Return values

- **HAL**: state

## 24.3 ETH Firmware driver defines

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

### 24.3.1 ETH

ETH

*ETH Address Aligned Beats*

`ETH_ADDRESSALIGNEDBEATS_ENABLE`

`ETH_ADDRESSALIGNEDBEATS_DISABLE`

*ETH Automatic Pad CRC Strip*

`ETH_AUTOMATICPADCRCSTRIP_ENABLE`

`ETH_AUTOMATICPADCRCSTRIP_DISABLE`

*ETH AutoNegotiation*

`ETH_AUTONEGOTIATION_ENABLE`

`ETH_AUTONEGOTIATION_DISABLE`

*ETH Back Off Limit*

`ETH_BACKOFFLIMIT_10`

`ETH_BACKOFFLIMIT_8`

`ETH_BACKOFFLIMIT_4`

`ETH_BACKOFFLIMIT_1`

*ETH Broadcast Frames Reception*

`ETH_BROADCASTFRAMESRECEPTION_ENABLE`

**ETH\_BROADCASTFRAMESRECEPTION\_DISABLE*****ETH Buffers setting*****ETH\_MAX\_PACKET\_SIZE**

ETH\_HEADER + ETH\_EXTRA + ETH\_VLAN\_TAG + ETH\_MAX\_ETH\_PAYLOAD + ETH\_CRC

**ETH\_HEADER**

6 byte Dest addr, 6 byte Src addr, 2 byte length/type

**ETH\_CRC**

Ethernet CRC

**ETH\_EXTRA**

Extra bytes in some cases

**ETH\_VLAN\_TAG**

optional 802.1q VLAN Tag

**ETH\_MIN\_ETH\_PAYLOAD**

Minimum Ethernet payload size

**ETH\_MAX\_ETH\_PAYLOAD**

Maximum Ethernet payload size

**ETH\_JUMBO\_FRAME\_PAYLOAD**

Jumbo frame payload size

**ETH\_RX\_BUF\_SIZE****ETH\_RXBUFNB****ETH\_TX\_BUF\_SIZE****ETH\_TXBUFNB*****ETH Carrier Sense*****ETH\_CARRIERSENCE\_ENABLE****ETH\_CARRIERSENCE\_DISABLE*****ETH Checksum Mode*****ETH\_CHECKSUM\_BY\_HARDWARE****ETH\_CHECKSUM\_BY\_SOFTWARE*****ETH Checksum Offload*****ETH\_CHECKSUMOFFLOAD\_ENABLE****ETH\_CHECKSUMOFFLOAD\_DISABLE*****ETH Deferral Check*****ETH\_DEFFERRALCHECK\_ENABLE****ETH\_DEFFERRALCHECK\_DISABLE*****ETH Destination Addr Filter***

ETH\_DESTINATIONADDRFILTER\_NORMAL

ETH\_DESTINATIONADDRFILTER\_INVERSE

***ETH DMA Arbitration***

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_1\_1

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_2\_1

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_3\_1

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_4\_1

ETH\_DMAARBITRATION\_RXPRIORTX

***ETH DMA Enhanced descriptor format***

ETH\_DMAENHANCEDDESCRIPTOR\_ENABLE

ETH\_DMAENHANCEDDESCRIPTOR\_DISABLE

***ETH DMA Flags***

ETH\_DMA\_FLAG\_TST

Time-stamp trigger interrupt (on DMA)

ETH\_DMA\_FLAG\_PMT

PMT interrupt (on DMA)

ETH\_DMA\_FLAG\_MMC

MMC interrupt (on DMA)

ETH\_DMA\_FLAG\_DATATRANSFERERROR

Error bits 0-Rx DMA, 1-Tx DMA

ETH\_DMA\_FLAG\_READWRITEERROR

Error bits 0-write transfer, 1-read transfer

ETH\_DMA\_FLAG\_ACCESSERROR

Error bits 0-data buffer, 1-desc. access

ETH\_DMA\_FLAG\_NIS

Normal interrupt summary flag

ETH\_DMA\_FLAG\_AIS

Abnormal interrupt summary flag

ETH\_DMA\_FLAG\_ER

Early receive flag

ETH\_DMA\_FLAG\_FBE

Fatal bus error flag

ETH\_DMA\_FLAG\_ET

Early transmit flag

ETH\_DMA\_FLAG\_RWT

Receive watchdog timeout flag

**ETH\_DMA\_FLAG\_RPS**

Receive process stopped flag

**ETH\_DMA\_FLAG\_RBU**

Receive buffer unavailable flag

**ETH\_DMA\_FLAG\_R**

Receive flag

**ETH\_DMA\_FLAG\_TU**

Underflow flag

**ETH\_DMA\_FLAG\_RO**

Overflow flag

**ETH\_DMA\_FLAG\_TJT**

Transmit jabber timeout flag

**ETH\_DMA\_FLAG\_TBU**

Transmit buffer unavailable flag

**ETH\_DMA\_FLAG\_TPS**

Transmit process stopped flag

**ETH\_DMA\_FLAG\_T**

Transmit flag

***ETH DMA Interrupts*****ETH\_DMA\_IT\_TST**

Time-stamp trigger interrupt (on DMA)

**ETH\_DMA\_IT\_PMT**

PMT interrupt (on DMA)

**ETH\_DMA\_IT\_MMC**

MMC interrupt (on DMA)

**ETH\_DMA\_IT\_NIS**

Normal interrupt summary

**ETH\_DMA\_IT\_AIS**

Abnormal interrupt summary

**ETH\_DMA\_IT\_ER**

Early receive interrupt

**ETH\_DMA\_IT\_FBE**

Fatal bus error interrupt

**ETH\_DMA\_IT\_ET**

Early transmit interrupt

**ETH\_DMA\_IT\_RWT**

Receive watchdog timeout interrupt

**ETH\_DMA\_IT\_RPS**

Receive process stopped interrupt



**ETH\_DMA\_IT\_RBU**

Receive buffer unavailable interrupt

**ETH\_DMA\_IT\_R**

Receive interrupt

**ETH\_DMA\_IT\_TU**

Underflow interrupt

**ETH\_DMA\_IT\_RO**

Overflow interrupt

**ETH\_DMA\_IT\_TJT**

Transmit jabber timeout interrupt

**ETH\_DMA\_IT\_TBU**

Transmit buffer unavailable interrupt

**ETH\_DMA\_IT\_TPS**

Transmit process stopped interrupt

**ETH\_DMA\_IT\_T**

Transmit interrupt

***ETH DMA overflow***

**ETH\_DMA\_OVERFLOW\_RXFIFOCOUNTER**

Overflow bit for FIFO overflow counter

**ETH\_DMA\_OVERFLOW\_MISSEDFRAMECOUNTER**

Overflow bit for missed frame counter

***ETH DMA receive process state***

**ETH\_DMA\_RECEIVEPROCESS\_STOPPED**

Stopped - Reset or Stop Rx Command issued

**ETH\_DMA\_RECEIVEPROCESS\_FETCHING**

Running - fetching the Rx descriptor

**ETH\_DMA\_RECEIVEPROCESS\_WAITING**

Running - waiting for packet

**ETH\_DMA\_RECEIVEPROCESS\_SUSPENDED**

Suspended - Rx Descriptor unavailable

**ETH\_DMA\_RECEIVEPROCESS\_CLOSING**

Running - closing descriptor

**ETH\_DMA\_RECEIVEPROCESS\_QUEUING**

Running - queuing the receive frame into host memory

***ETH DMA RX Descriptor***

**ETH\_DMARXDESC\_OWN**

OWN bit: descriptor is owned by DMA engine

**ETH\_DMARXDESC\_AFM**

DA Filter Fail for the rx frame

**ETH\_DMARXDESC\_FL**

Receive descriptor frame length

**ETH\_DMARXDESC\_ES**

Error summary: OR of the following bits: DE || OE || IPC || LC || RWT || RE || CE

**ETH\_DMARXDESC\_DE**

Descriptor error: no more descriptors for receive frame

**ETH\_DMARXDESC\_SAF**

SA Filter Fail for the received frame

**ETH\_DMARXDESC\_LE**

Frame size not matching with length field

**ETH\_DMARXDESC\_OE**

Overflow Error: Frame was damaged due to buffer overflow

**ETH\_DMARXDESC\_VLAN**

VLAN Tag: received frame is a VLAN frame

**ETH\_DMARXDESC\_FS**

First descriptor of the frame

**ETH\_DMARXDESC\_LS**

Last descriptor of the frame

**ETH\_DMARXDESC\_IPV4HCE**

IPC Checksum Error: Rx Ipv4 header checksum error

**ETH\_DMARXDESC\_LC**

Late collision occurred during reception

**ETH\_DMARXDESC\_FT**

Frame type - Ethernet, otherwise 802.3

**ETH\_DMARXDESC\_RWT**

Receive Watchdog Timeout: watchdog timer expired during reception

**ETH\_DMARXDESC\_RE**

Receive error: error reported by MII interface

**ETH\_DMARXDESC\_DBE**

Dribble bit error: frame contains non int multiple of 8 bits

**ETH\_DMARXDESC\_CE**

CRC error

**ETH\_DMARXDESC\_MAMPCE**

Rx MAC Address/Payload Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error

**ETH\_DMARXDESC\_DIC**

Disable Interrupt on Completion

**ETH\_DMARXDESC\_RBS2**

Receive Buffer2 Size

**ETH\_DMARXDESC\_RER**  
Receive End of Ring

**ETH\_DMARXDESC\_RCH**  
Second Address Chained

**ETH\_DMARXDESC\_RBS1**  
Receive Buffer1 Size

**ETH\_DMARXDESC\_B1AP**  
Buffer1 Address Pointer

**ETH\_DMARXDESC\_B2AP**  
Buffer2 Address Pointer

**ETH\_DMAPTPRXDESC\_PTPV**

**ETH\_DMAPTPRXDESC\_PTPFT**

**ETH\_DMAPTPRXDESC\_PTPMT**

**ETH\_DMAPTPRXDESC\_PTPMT\_SYNC**

**ETH\_DMAPTPRXDESC\_PTPMT\_FOLLOWUP**

**ETH\_DMAPTPRXDESC\_PTPMT\_DELAYREQ**

**ETH\_DMAPTPRXDESC\_PTPMT\_DELAYRESP**

**ETH\_DMAPTPRXDESC\_PTPMT\_PDELAYREQ\_ANNOUNCE**

**ETH\_DMAPTPRXDESC\_PTPMT\_PDELAYRESP\_MANAG**

**ETH\_DMAPTPRXDESC\_PTPMT\_PDELAYRESPFOLLOWUP\_SIGNAL**

**ETH\_DMAPTPRXDESC\_IPV6PR**

**ETH\_DMAPTPRXDESC\_IPV4PR**

**ETH\_DMAPTPRXDESC\_IPCB**

**ETH\_DMAPTPRXDESC\_IPPE**

**ETH\_DMAPTPRXDESC\_IPHE**

**ETH\_DMAPTPRXDESC\_IPPT**

**ETH\_DMAPTPRXDESC\_IPPT\_UDP**

**ETH\_DMAPTPRXDESC\_IPPT\_TCP**

**ETH\_DMAPTPRXDESC\_IPPT\_ICMP**

**ETH\_DMAPTPRXDESC\_RTSL**

**ETH\_DMAPTPRXDESC\_RTSH**

**ETH DMA Rx descriptor buffers****ETH\_DMARXDESC\_BUFFER1**

DMA Rx Desc Buffer1

**ETH\_DMARXDESC\_BUFFER2**

DMA Rx Desc Buffer2

**ETH DMA transmit process state****ETH\_DMA\_TRANSMITPROCESS\_STOPPED**

Stopped - Reset or Stop Tx Command issued

**ETH\_DMA\_TRANSMITPROCESS\_FETCHING**

Running - fetching the Tx descriptor

**ETH\_DMA\_TRANSMITPROCESS\_WAITING**

Running - waiting for status

**ETH\_DMA\_TRANSMITPROCESS\_READING**

Running - reading the data from host memory

**ETH\_DMA\_TRANSMITPROCESS\_SUSPENDED**

Suspended - Tx Descriptor unavailable

**ETH\_DMA\_TRANSMITPROCESS\_CLOSING**

Running - closing Rx descriptor

**ETH DMA TX Descriptor****ETH\_DMATXDESC\_OWN**

OWN bit: descriptor is owned by DMA engine

**ETH\_DMATXDESC\_IC**

Interrupt on Completion

**ETH\_DMATXDESC\_LS**

Last Segment

**ETH\_DMATXDESC\_FS**

First Segment

**ETH\_DMATXDESC\_DC**

Disable CRC

**ETH\_DMATXDESC\_DP**

Disable Padding

**ETH\_DMATXDESC\_TTSE**

Transmit Time Stamp Enable

**ETH\_DMATXDESC\_CIC**

Checksum Insertion Control: 4 cases

**ETH\_DMATXDESC\_CIC\_BYPASS**

Do Nothing: Checksum Engine is bypassed

**ETH\_DMATXDESC\_CIC\_IPV4HEADER**

IPV4 header Checksum Insertion

**ETH\_DMATXDESC\_CIC\_TCPUDPICMP\_SEGMENT**

TCP/UDP/ICMP Checksum Insertion calculated over segment only

**ETH\_DMATXDESC\_CIC\_TCPUDPICMP\_FULL**

TCP/UDP/ICMP Checksum Insertion fully calculated

**ETH\_DMATXDESC\_TER**

Transmit End of Ring

**ETH\_DMATXDESC\_TCH**

Second Address Chained

**ETH\_DMATXDESC\_TTSS**

Tx Time Stamp Status

**ETH\_DMATXDESC\_IHE**

IP Header Error

**ETH\_DMATXDESC\_ES**

Error summary: OR of the following bits: UE || ED || EC || LCO || NC || LCA || FF || JT

**ETH\_DMATXDESC\_JT**

Jabber Timeout

**ETH\_DMATXDESC\_FF**

Frame Flushed: DMA/MTL flushed the frame due to SW flush

**ETH\_DMATXDESC\_PCE**

Payload Checksum Error

**ETH\_DMATXDESC\_LCA**

Loss of Carrier: carrier lost during transmission

**ETH\_DMATXDESC\_NC**

No Carrier: no carrier signal from the transceiver

**ETH\_DMATXDESC\_LCO**

Late Collision: transmission aborted due to collision

**ETH\_DMATXDESC\_EC**

Excessive Collision: transmission aborted after 16 collisions

**ETH\_DMATXDESC\_VF**

VLAN Frame

**ETH\_DMATXDESC\_CC**

Collision Count

**ETH\_DMATXDESC\_ED**

Excessive Deferral

**ETH\_DMATXDESC\_UF**

Underflow Error: late data arrival from the memory

**ETH\_DMATXDESC\_DB**

Deferred Bit

**ETH\_DMATXDESC\_TBS2**

Transmit Buffer2 Size

**ETH\_DMATXDESC\_TBS1**

Transmit Buffer1 Size

**ETH\_DMATXDESC\_B1AP**

Buffer1 Address Pointer

**ETH\_DMATXDESC\_B2AP**

Buffer2 Address Pointer

**ETH\_DMAPTPTXDESC\_TTSL**

**ETH\_DMAPTPTXDESC\_TTSH**

***ETH DMA Tx descriptor Checksum Insertion Control***

**ETH\_DMATXDESC\_CHECKSUMBYPASS**

Checksum engine bypass

**ETH\_DMATXDESC\_CHECKSUMIPV4HEADER**

IPv4 header checksum insertion

**ETH\_DMATXDESC\_CHECKSUMTCPUDPICMPSEGMENT**

TCP/UDP/ICMP checksum insertion. Pseudo header checksum is assumed to be present

**ETH\_DMATXDESC\_CHECKSUMTCPUDPICMPFULL**

TCP/UDP/ICMP checksum fully in hardware including pseudo header

***ETH DMA Tx descriptor segment***

**ETH\_DMATXDESC\_LASTSEGMENTS**

Last Segment

**ETH\_DMATXDESC\_FIRSTSEGMENT**

First Segment

***ETH Drop TCP IP Checksum Error Frame***

**ETH\_DROPTCPIPChecksumErrorFrame\_ENABLE**

**ETH\_DROPTCPIPChecksumErrorFrame\_DISABLE**

***ETH Duplex Mode***

**ETH\_MODE\_FULLDUPLEX**

**ETH\_MODE\_HALFDUPLEX**

***ETH Exported Macros***

**\_\_HAL\_ETH\_RESET\_HANDLE\_STATE**

**Description:**

- Reset ETH handle state.

**Parameters:**

- `__HANDLE__`: specifies the ETH handle.

**Return value:**

- None

### **\_\_HAL\_ETH\_DMATXDESC\_GET\_FLAG**

**Description:**

- Checks whether the specified ETHERNET DMA Tx Desc flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__FLAG__`: specifies the flag of TDES0 to check.

**Return value:**

- the: ETH\_DMA Tx Desc Flag (SET or RESET).

### **\_\_HAL\_ETH\_DMARXDESC\_GET\_FLAG**

**Description:**

- Checks whether the specified ETHERNET DMA Rx Desc flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__FLAG__`: specifies the flag of RDES0 to check.

**Return value:**

- the: ETH\_DMA Tx Desc Flag (SET or RESET).

### **\_\_HAL\_ETH\_DMARXDESC\_ENABLE\_IT**

**Description:**

- Enables the specified DMA Rx Desc receive interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

### **\_\_HAL\_ETH\_DMARXDESC\_DISABLE\_IT**

**Description:**

- Disables the specified DMA Rx Desc receive interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

### **\_\_HAL\_ETH\_DMARXDESC\_SET\_OWN\_BIT**

**Description:**

- Set the specified DMA Rx Desc Own bit.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

### **\_\_HAL\_ETH\_DMATXDESC\_GET\_COLLISION\_COUNT**

**Description:**

- Returns the specified ETHERNET DMA Tx Desc collision count.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- The: Transmit descriptor collision counter value.

#### \_\_HAL\_ETH\_DMATXDESC\_SET\_OWN\_BIT

**Description:**

- Set the specified DMA Tx Desc Own bit.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### \_\_HAL\_ETH\_DMATXDESC\_ENABLE\_IT

**Description:**

- Enables the specified DMA Tx Desc Transmit interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### \_\_HAL\_ETH\_DMATXDESC\_DISABLE\_IT

**Description:**

- Disables the specified DMA Tx Desc Transmit interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### \_\_HAL\_ETH\_DMATXDESC\_CHECKSUM\_INSERTION

**Description:**

- Selects the specified ETHERNET DMA Tx Desc Checksum Insertion.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__CHECKSUM__`: specifies is the DMA Tx desc checksum insertion. This parameter can be one of the following values:
  - `ETH_DMATXDESC_CHECKSUMBYPASS` : Checksum bypass
  - `ETH_DMATXDESC_CHECKSUMIPV4HEADER` : IPv4 header checksum
  - `ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT` : TCP/UDP/ICMP checksum. Pseudo header checksum is assumed to be present
  - `ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL` : TCP/UDP/ICMP checksum fully in hardware including pseudo header

**Return value:**

- None

#### \_\_HAL\_ETH\_DMATXDESC\_CRC\_ENABLE

**Description:**

- Enables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None



#### \_\_HAL\_ETH\_DMATXDESC\_CRC\_DISABLE

**Description:**

- Disables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### \_\_HAL\_ETH\_DMATXDESC\_SHORT\_FRAME\_PADDING\_ENABLE

**Description:**

- Enables the DMA Tx Desc padding for frame shorter than 64 bytes.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### \_\_HAL\_ETH\_DMATXDESC\_SHORT\_FRAME\_PADDING\_DISABLE

**Description:**

- Disables the DMA Tx Desc padding for frame shorter than 64 bytes.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### \_\_HAL\_ETH\_MAC\_ENABLE\_IT

**Description:**

- Enables the specified ETHERNET MAC interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `ETH_MAC_IT_TST` : Time stamp trigger interrupt
  - `ETH_MAC_IT_PMT` : PMT interrupt

**Return value:**

- None

#### \_\_HAL\_ETH\_MAC\_DISABLE\_IT

**Description:**

- Disables the specified ETHERNET MAC interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `ETH_MAC_IT_TST` : Time stamp trigger interrupt
  - `ETH_MAC_IT_PMT` : PMT interrupt

**Return value:**

- None

#### **\_\_HAL\_ETH\_INITIATE\_PAUSE\_CONTROL\_FRAME**

**Description:**

- Initiate a Pause Control Frame (Full-duplex only).

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### **\_\_HAL\_ETH\_GET\_FLOW\_CONTROL\_BUSY\_STATUS**

**Description:**

- Checks whether the ETHERNET flow control busy bit is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- The: new state of flow control busy status bit (SET or RESET).

#### **\_\_HAL\_ETH\_BACK\_PRESSURE\_ACTIVATION\_ENABLE**

**Description:**

- Enables the MAC Back Pressure operation activation (Half-duplex only).

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### **\_\_HAL\_ETH\_BACK\_PRESSURE\_ACTIVATION\_DISABLE**

**Description:**

- Disables the MAC BackPressure operation activation (Half-duplex only).

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### **\_\_HAL\_ETH\_MAC\_GET\_FLAG**

**Description:**

- Checks whether the specified ETHERNET MAC flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `ETH_MAC_FLAG_TST` : Time stamp trigger flag
  - `ETH_MAC_FLAG_MMCT` : MMC transmit flag
  - `ETH_MAC_FLAG_MMCR` : MMC receive flag
  - `ETH_MAC_FLAG_MMC` : MMC flag
  - `ETH_MAC_FLAG_PMT` : PMT flag

**Return value:**

- The: state of ETHERNET MAC flag.

### `__HAL_ETH_DMA_ENABLE_IT`

**Description:**

- Enables the specified ETHERNET DMA interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET DMA interrupt sources to be enabled

**Return value:**

- None

### `__HAL_ETH_DMA_DISABLE_IT`

**Description:**

- Disables the specified ETHERNET DMA interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET DMA interrupt sources to be disabled.

**Return value:**

- None

### `__HAL_ETH_DMA_CLEAR_IT`

**Description:**

- Clears the ETHERNET DMA IT pending bit.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear.

**Return value:**

- None

### `__HAL_ETH_DMA_GET_FLAG`

**Description:**

- Checks whether the specified ETHERNET DMA flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__FLAG__`: specifies the flag to check.

**Return value:**

- The: new state of ETH\_DMA\_FLAG (SET or RESET).

### `__HAL_ETH_DMA_CLEAR_FLAG`

**Description:**

- Checks whether the specified ETHERNET DMA flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__FLAG__`: specifies the flag to clear.

**Return value:**

- The: new state of ETH\_DMA\_FLAG (SET or RESET).

### **\_\_HAL\_ETH\_GET\_DMA\_OVERFLOW\_STATUS**

**Description:**

- Checks whether the specified ETHERNET DMA overflow flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__OVERFLOW__`: specifies the DMA overflow flag to check. This parameter can be one of the following values:
  - `ETH_DMA_OVERFLOW_RXFIFOCOUNTER` : Overflow for FIFO Overflows Counter
  - `ETH_DMA_OVERFLOW_MISSEDFRAMECOUNTER` : Overflow for Buffer Unavailable Missed Frame Counter

**Return value:**

- The: state of ETHERNET DMA overflow Flag (SET or RESET).

### **\_\_HAL\_ETH\_SET\_RECEIVE\_WATCHDOG\_TIMER**

**Description:**

- Set the DMA Receive status watchdog timer register value.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__VALUE__`: DMA Receive status watchdog timer register value

**Return value:**

- None

### **\_\_HAL\_ETH\_GLOBAL\_UNICAST\_WAKEUP\_ENABLE**

**Description:**

- Enables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

### **\_\_HAL\_ETH\_GLOBAL\_UNICAST\_WAKEUP\_DISABLE**

**Description:**

- Disables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

### **\_\_HAL\_ETH\_WAKEUP\_FRAME\_DETECTION\_ENABLE**

**Description:**

- Enables the MAC Wake-Up Frame Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

#### **\_\_HAL\_ETH\_WAKEUP\_FRAME\_DETECTION\_DISABLE**

**Description:**

- Disables the MAC Wake-Up Frame Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

#### **\_\_HAL\_ETH\_MAGIC\_PACKET\_DETECTION\_ENABLE**

**Description:**

- Enables the MAC Magic Packet Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

#### **\_\_HAL\_ETH\_MAGIC\_PACKET\_DETECTION\_DISABLE**

**Description:**

- Disables the MAC Magic Packet Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

#### **\_\_HAL\_ETH\_POWER\_DOWN\_ENABLE**

**Description:**

- Enables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

#### **\_\_HAL\_ETH\_POWER\_DOWN\_DISABLE**

**Description:**

- Disables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

### \_\_HAL\_ETH\_GET\_PMT\_FLAG\_STATUS

**Description:**

- Checks whether the specified ETHERNET PMT flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `ETH_PMT_FLAG_WUFFRPR` : Wake-Up Frame Filter Register Pointer Reset
  - `ETH_PMT_FLAG_WUFR` : Wake-Up Frame Received
  - `ETH_PMT_FLAG_MPR` : Magic Packet Received

**Return value:**

- The: new state of ETHERNET PMT Flag (SET or RESET).

### \_\_HAL\_ETH\_MMC\_COUNTER\_FULL\_PRESET

**Description:**

- Preset and Initialize the MMC counters to almost-full value: `0xFFFF_FFF0` (full - 16)

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

### \_\_HAL\_ETH\_MMC\_COUNTER\_HALF\_PRESET

**Description:**

- Preset and Initialize the MMC counters to almost-half value: `0x7FFF_FFF0` (half - 16)

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

### \_\_HAL\_ETH\_MMC\_COUNTER\_FREEZE\_ENABLE

**Description:**

- Enables the MMC Counter Freeze.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

### \_\_HAL\_ETH\_MMC\_COUNTER\_FREEZE\_DISABLE

**Description:**

- Disables the MMC Counter Freeze.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**\_\_HAL\_ETH\_ETH\_MMC\_RESET\_ONREAD\_ENABLE****Description:**

- Enables the MMC Reset On Read.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**\_\_HAL\_ETH\_ETH\_MMC\_RESET\_ONREAD\_DISABLE****Description:**

- Disables the MMC Reset On Read.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**\_\_HAL\_ETH\_ETH\_MMC\_COUNTER\_ROLLOVER\_ENABLE****Description:**

- Enables the MMC Counter Stop Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**\_\_HAL\_ETH\_ETH\_MMC\_COUNTER\_ROLLOVER\_DISABLE****Description:**

- Disables the MMC Counter Stop Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**\_\_HAL\_ETH\_MMC\_COUNTERS\_RESET****Description:**

- Resets the MMC Counters.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

### `__HAL_ETH_MMC_RX_IT_ENABLE`

**Description:**

- Enables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__INTERRUPT__`: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - `ETH_MMC_IT_RGUF` : When Rx good unicast frames counter reaches half the maximum value
  - `ETH_MMC_IT_RFAE` : When Rx alignment error counter reaches half the maximum value
  - `ETH_MMC_IT_RFCE` : When Rx crc error counter reaches half the maximum value

**Return value:**

- None

### `__HAL_ETH_MMC_RX_IT_DISABLE`

**Description:**

- Disables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__INTERRUPT__`: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - `ETH_MMC_IT_RGUF` : When Rx good unicast frames counter reaches half the maximum value
  - `ETH_MMC_IT_RFAE` : When Rx alignment error counter reaches half the maximum value
  - `ETH_MMC_IT_RFCE` : When Rx crc error counter reaches half the maximum value

**Return value:**

- None

### `__HAL_ETH_MMC_TX_IT_ENABLE`

**Description:**

- Enables the specified ETHERNET MMC Tx interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__INTERRUPT__`: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - `ETH_MMC_IT_TGF` : When Tx good frame counter reaches half the maximum value
  - `ETH_MMC_IT_TGFMSC`: When Tx good multi col counter reaches half the maximum value
  - `ETH_MMC_IT_TGFSC` : When Tx good single col counter reaches half the maximum value

**Return value:**

- None



### `__HAL_ETH_MMC_TX_IT_DISABLE`

**Description:**

- Disables the specified ETHERNET MMC Tx interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__INTERRUPT__`: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - `ETH_MMC_IT_TGF` : When Tx good frame counter reaches half the maximum value
  - `ETH_MMC_IT_TGFMSC`: When Tx good multi col counter reaches half the maximum value
  - `ETH_MMC_IT_TGFSC` : When Tx good single col counter reaches half the maximum value

**Return value:**

- None

### `__HAL_ETH_WAKEUP_EXTI_ENABLE_IT`

**Description:**

- Enables the ETH External interrupt line.

**Return value:**

- None

### `__HAL_ETH_WAKEUP_EXTI_DISABLE_IT`

**Description:**

- Disables the ETH External interrupt line.

**Return value:**

- None

### `__HAL_ETH_WAKEUP_EXTI_ENABLE_EVENT`

**Description:**

- Enable event on ETH External event line.

**Return value:**

- None.

### `__HAL_ETH_WAKEUP_EXTI_DISABLE_EVENT`

**Description:**

- Disable event on ETH External event line.

**Return value:**

- None.

### `__HAL_ETH_WAKEUP_EXTI_GET_FLAG`

**Description:**

- Get flag of the ETH External interrupt line.

**Return value:**

- None

### `__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG`

**Description:**

- Clear flag of the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_ENABLE\_RISING\_EDGE\_TRIGGER

**Description:**

- Enables rising edge trigger to the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_DISABLE\_RISING\_EDGE\_TRIGGER

**Description:**

- Disables the rising edge trigger to the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_ENABLE\_FALLING\_EDGE\_TRIGGER

**Description:**

- Enables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_DISABLE\_FALLING\_EDGE\_TRIGGER

**Description:**

- Disables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_ENABLE\_FALLINGRISING\_TRIGGER

**Description:**

- Enables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_DISABLE\_FALLINGRISING\_TRIGGER

**Description:**

- Disables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

#### \_\_HAL\_ETH\_WAKEUP\_EXTI\_GENERATE\_SWIT

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.

**ETH EXTI LINE WAKEUP**

#### ETH\_EXTI\_LINE\_WAKEUP

External interrupt line 19 Connected to the ETH EXTI Line

**ETH Fixed Burst**

#### ETH\_FIXEDBURST\_ENABLE

#### ETH\_FIXEDBURST\_DISABLE

**ETH Flush Received Frame**

ETH\_FLUSHRECEIVEDFRAME\_ENABLE

ETH\_FLUSHRECEIVEDFRAME\_DISABLE

***ETH Forward Error Frames***

ETH\_FORWARDERRORFRAMES\_ENABLE

ETH\_FORWARDERRORFRAMES\_DISABLE

***ETH Forward Undersized Good Frames***

ETH\_FORWARDUNDERSIZEDGOODFRAMES\_ENABLE

ETH\_FORWARDUNDERSIZEDGOODFRAMES\_DISABLE

***ETH Inter Frame Gap***

ETH\_INTERFRAMEGAP\_96BIT

minimum IFG between frames during transmission is 96Bit

ETH\_INTERFRAMEGAP\_88BIT

minimum IFG between frames during transmission is 88Bit

ETH\_INTERFRAMEGAP\_80BIT

minimum IFG between frames during transmission is 80Bit

ETH\_INTERFRAMEGAP\_72BIT

minimum IFG between frames during transmission is 72Bit

ETH\_INTERFRAMEGAP\_64BIT

minimum IFG between frames during transmission is 64Bit

ETH\_INTERFRAMEGAP\_56BIT

minimum IFG between frames during transmission is 56Bit

ETH\_INTERFRAMEGAP\_48BIT

minimum IFG between frames during transmission is 48Bit

ETH\_INTERFRAMEGAP\_40BIT

minimum IFG between frames during transmission is 40Bit

***ETH Jabber***

ETH\_JABBER\_ENABLE

ETH\_JABBER\_DISABLE

***ETH Loop Back Mode***

ETH\_LOOPBACKMODE\_ENABLE

ETH\_LOOPBACKMODE\_DISABLE

***ETH MAC addresses***

ETH\_MAC\_ADDRESS0

ETH\_MAC\_ADDRESS1

ETH\_MAC\_ADDRESS2

## ETH\_MAC\_ADDRESS3

***ETH MAC addresses filter Mask bytes***

### ETH\_MAC\_ADDRESSMASK\_BYTE6

Mask MAC Address high reg bits [15:8]

### ETH\_MAC\_ADDRESSMASK\_BYTE5

Mask MAC Address high reg bits [7:0]

### ETH\_MAC\_ADDRESSMASK\_BYTE4

Mask MAC Address low reg bits [31:24]

### ETH\_MAC\_ADDRESSMASK\_BYTE3

Mask MAC Address low reg bits [23:16]

### ETH\_MAC\_ADDRESSMASK\_BYTE2

Mask MAC Address low reg bits [15:8]

### ETH\_MAC\_ADDRESSMASK\_BYTE1

Mask MAC Address low reg bits [7:0]

***ETH MAC addresses filter SA DA***

## ETH\_MAC\_ADDRESSFILTER\_SA

## ETH\_MAC\_ADDRESSFILTER\_DA

***ETH MAC Flags***

### ETH\_MAC\_FLAG\_TST

Time stamp trigger flag (on MAC)

### ETH\_MAC\_FLAG\_MMCT

MMC transmit flag

### ETH\_MAC\_FLAG\_MMCR

MMC receive flag

### ETH\_MAC\_FLAG\_MMC

MMC flag (on MAC)

### ETH\_MAC\_FLAG\_PMT

PMT flag (on MAC)

***ETH MAC Interrupts***

## ETH\_MAC\_IT\_TST

Time stamp trigger interrupt (on MAC)

## ETH\_MAC\_IT\_MMCT

MMC transmit interrupt

## ETH\_MAC\_IT\_MMCR

MMC receive interrupt

## ETH\_MAC\_IT\_MMC

MMC interrupt (on MAC)

## ETH\_MAC\_IT\_PMT

PMT interrupt (on MAC)

***ETH Media Interface***

ETH\_MEDIA\_INTERFACE\_MII

ETH\_MEDIA\_INTERFACE\_RMII

***ETH MMC Rx Interrupts***

ETH\_MMC\_IT\_RGUF

When Rx good unicast frames counter reaches half the maximum value

ETH\_MMC\_IT\_RFAE

When Rx alignment error counter reaches half the maximum value

ETH\_MMC\_IT\_RFCE

When Rx crc error counter reaches half the maximum value

***ETH MMC Tx Interrupts***

ETH\_MMC\_IT\_TGF

When Tx good frame counter reaches half the maximum value

ETH\_MMC\_IT\_TGFMSC

When Tx good multi col counter reaches half the maximum value

ETH\_MMC\_IT\_TGFSC

When Tx good single col counter reaches half the maximum value

***ETH Multicast Frames Filter***

ETH\_MULTICASTFRAMESFILTER\_PERFECTHASHTABLE

ETH\_MULTICASTFRAMESFILTER\_HASHTABLE

ETH\_MULTICASTFRAMESFILTER\_PERFECT

ETH\_MULTICASTFRAMESFILTER\_NONE

***ETH Pass Control Frames***

ETH\_PASSCONTROLFRAMES\_BLOCKALL

MAC filters all control frames from reaching the application

ETH\_PASSCONTROLFRAMES\_FORWARDALL

MAC forwards all control frames to application even if they fail the Address Filter

ETH\_PASSCONTROLFRAMES\_FORWARDPASSEDADDRFILTER

MAC forwards control frames that pass the Address Filter.

***ETH Pause Low Threshold***

ETH\_PAUSELOWTHRESHOLD\_MINUS4

Pause time minus 4 slot times

ETH\_PAUSELOWTHRESHOLD\_MINUS28

Pause time minus 28 slot times

ETH\_PAUSELOWTHRESHOLD\_MINUS144

Pause time minus 144 slot times

ETH\_PAUSELOWTHRESHOLD\_MINUS256

Pause time minus 256 slot times

**ETH PMT Flags**

**ETH\_PMT\_FLAG\_WUFFRPR**

Wake-Up Frame Filter Register Pointer Reset

**ETH\_PMT\_FLAG\_WUFR**

Wake-Up Frame Received

**ETH\_PMT\_FLAG\_MPR**

Magic Packet Received

**ETH Promiscuous Mode**

**ETH\_PROMISCUOUS\_MODE\_ENABLE**

**ETH\_PROMISCUOUS\_MODE\_DISABLE**

**ETH Receive All**

**ETH\_RECEIVEALL\_ENABLE**

**ETH\_RECEIVEALL\_DISABLE**

**ETH Receive Flow Control**

**ETH\_RECEIVEFLOWCONTROL\_ENABLE**

**ETH\_RECEIVEFLOWCONTROL\_DISABLE**

**ETH Receive Own**

**ETH\_RECEIVEOWN\_ENABLE**

**ETH\_RECEIVEOWN\_DISABLE**

**ETH Receive Store Forward**

**ETH\_RECEIVESTOREFORWARD\_ENABLE**

**ETH\_RECEIVESTOREFORWARD\_DISABLE**

**ETH Receive Threshold Control**

**ETH\_RECEIVEDTHRESHOLDCONTROL\_64BYTES**

threshold level of the MTL Receive FIFO is 64 Bytes

**ETH\_RECEIVEDTHRESHOLDCONTROL\_32BYTES**

threshold level of the MTL Receive FIFO is 32 Bytes

**ETH\_RECEIVEDTHRESHOLDCONTROL\_96BYTES**

threshold level of the MTL Receive FIFO is 96 Bytes

**ETH\_RECEIVEDTHRESHOLDCONTROL\_128BYTES**

threshold level of the MTL Receive FIFO is 128 Bytes

**ETH Retry Transmission**

**ETH\_RETRYTRANSMISSION\_ENABLE**

**ETH\_RETRYTRANSMISSION\_DISABLE**

**ETH Rx DMA Burst Length**

**ETH\_RXDMABURSTLENGTH\_1BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 1

**ETH\_RXDMABURSTLENGTH\_2BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 2

**ETH\_RXDMABURSTLENGTH\_4BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 4

**ETH\_RXDMABURSTLENGTH\_8BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 8

**ETH\_RXDMABURSTLENGTH\_16BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 16

**ETH\_RXDMABURSTLENGTH\_32BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 32

**ETH\_RXDMABURSTLENGTH\_4XPBL\_4BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 4

**ETH\_RXDMABURSTLENGTH\_4XPBL\_8BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 8

**ETH\_RXDMABURSTLENGTH\_4XPBL\_16BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 16

**ETH\_RXDMABURSTLENGTH\_4XPBL\_32BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 32

**ETH\_RXDMABURSTLENGTH\_4XPBL\_64BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 64

**ETH\_RXDMABURSTLENGTH\_4XPBL\_128BEAT**

maximum number of beats to be transferred in one RxDMA transaction is 128

***ETH Rx Mode*****ETH\_RXPOLLING\_MODE****ETH\_RXINTERRUPT\_MODE*****ETH Second Frame Operate*****ETH\_SECONDFRAMEOPERARTE\_ENABLE****ETH\_SECONDFRAMEOPERARTE\_DISABLE*****ETH Source Addr Filter*****ETH\_SOURCEADDRFILTER\_NORMAL\_ENABLE****ETH\_SOURCEADDRFILTER\_INVERSE\_ENABLE****ETH\_SOURCEADDRFILTER\_DISABLE*****ETH Speed*****ETH\_SPEED\_10M****ETH\_SPEED\_100M**

**ETH Transmit Flow Control****ETH\_TRANSMITFLOWCONTROL\_ENABLE****ETH\_TRANSMITFLOWCONTROL\_DISABLE****ETH Transmit Store Forward****ETH\_TRANSMITSTOREFORWARD\_ENABLE****ETH\_TRANSMITSTOREFORWARD\_DISABLE****ETH Transmit Threshold Control****ETH\_TRANSMITTHRESHOLDCONTROL\_64BYTES**

threshold level of the MTL Transmit FIFO is 64 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_128BYTES**

threshold level of the MTL Transmit FIFO is 128 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_192BYTES**

threshold level of the MTL Transmit FIFO is 192 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_256BYTES**

threshold level of the MTL Transmit FIFO is 256 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_40BYTES**

threshold level of the MTL Transmit FIFO is 40 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_32BYTES**

threshold level of the MTL Transmit FIFO is 32 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_24BYTES**

threshold level of the MTL Transmit FIFO is 24 Bytes

**ETH\_TRANSMITTHRESHOLDCONTROL\_16BYTES**

threshold level of the MTL Transmit FIFO is 16 Bytes

**ETH Tx DMA Burst Length****ETH\_TXDMABURSTLENGTH\_1BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 1

**ETH\_TXDMABURSTLENGTH\_2BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 2

**ETH\_TXDMABURSTLENGTH\_4BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 4

**ETH\_TXDMABURSTLENGTH\_8BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 8

**ETH\_TXDMABURSTLENGTH\_16BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 16

**ETH\_TXDMABURSTLENGTH\_32BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 32

**ETH\_TXDMABURSTLENGTH\_4XPBL\_4BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 4



**ETH\_TXDMABURSTLENGTH\_4XPBL\_8BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 8

**ETH\_TXDMABURSTLENGTH\_4XPBL\_16BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 16

**ETH\_TXDMABURSTLENGTH\_4XPBL\_32BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 32

**ETH\_TXDMABURSTLENGTH\_4XPBL\_64BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 64

**ETH\_TXDMABURSTLENGTH\_4XPBL\_128BEAT**

maximum number of beats to be transferred in one TxDMA (or both) transaction is 128

***ETH Unicast Frames Filter*****ETH\_UNICASTFRAMESFILTER\_PERFECTHASHTABLE****ETH\_UNICASTFRAMESFILTER\_HASHTABLE****ETH\_UNICASTFRAMESFILTER\_PERFECT*****ETH Unicast Pause Frame Detect*****ETH\_UNICASTPAUSEFRAMEDetect\_ENABLE****ETH\_UNICASTPAUSEFRAMEDetect\_DISABLE*****ETH VLAN Tag Comparison*****ETH\_VLANTAGCOMPARISON\_12BIT****ETH\_VLANTAGCOMPARISON\_16BIT*****ETH Watchdog*****ETH\_WATCHDOG\_ENABLE****ETH\_WATCHDOG\_DISABLE*****ETH Zero Quanta Pause*****ETH\_ZEROQUANTAPAUSE\_ENABLE****ETH\_ZEROQUANTAPAUSE\_DISABLE**

## 25 HAL EXTI Generic Driver

### 25.1 EXTI Firmware driver registers structures

#### 25.1.1 EXTI\_HandleTypeDef

*EXTI\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 25.1.2 EXTI\_ConfigTypeDef

*EXTI\_ConfigTypeDef* is defined in the `stm32f4xx_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](#)

### 25.2 EXTI Firmware driver API description

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

#### 25.2.1 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 interrupts 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.

### 25.2.2 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 "GPIOSeI" 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\_GetPending().
7. Generate software interrupt using HAL\_EXTI\_GenerateSWI().

### 25.2.3 Configuration functions

This section contains the following APIs:

- [\*HAL\\_EXTI\\_SetConfigLine\(\)\*](#)
- [\*HAL\\_EXTI\\_GetConfigLine\(\)\*](#)
- [\*HAL\\_EXTI\\_ClearConfigLine\(\)\*](#)
- [\*HAL\\_EXTI\\_RegisterCallback\(\)\*](#)
- [\*HAL\\_EXTI\\_GetHandle\(\)\*](#)

### 25.2.4 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) pPendingCbf)

#### 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.
- **pPendingCbf**: 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**: Specify which pending edge as to be checked. This parameter can be one of the following values:
  - EXTI\_TRIGGER\_RISING\_FALLING This parameter is kept for compatibility with other series.

### 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**: Specify which pending edge as to be clear. This parameter can be one of the following values:
  - EXTI\_TRIGGER\_RISING\_FALLING This parameter is kept for compatibility with other series.

### 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.:**

## 25.3 EXTI Firmware driver defines

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

### 25.3.1 EXTI

EXTI

*EXTI GPIOSeI*

EXTI\_GPIOA

EXTI\_GPIOB

EXTI\_GPIOC

EXTI\_GPIOD

EXTI\_GPIOE

EXTI\_GPIOF

EXTI\_GPIOG

EXTI\_GPIOH

EXTI\_GPIOI

EXTI\_GPIOJ

EXTI\_GPIOK

*EXTI Line*

EXTI\_LINE\_0

External interrupt line 0

EXTI\_LINE\_1

External interrupt line 1

EXTI\_LINE\_2

External interrupt line 2

EXTI\_LINE\_3

External interrupt line 3

EXTI\_LINE\_4

External interrupt line 4

EXTI\_LINE\_5

External interrupt line 5

---

<b>EXTI_LINE_6</b>	External interrupt line 6
<b>EXTI_LINE_7</b>	External interrupt line 7
<b>EXTI_LINE_8</b>	External interrupt line 8
<b>EXTI_LINE_9</b>	External interrupt line 9
<b>EXTI_LINE_10</b>	External interrupt line 10
<b>EXTI_LINE_11</b>	External interrupt line 11
<b>EXTI_LINE_12</b>	External interrupt line 12
<b>EXTI_LINE_13</b>	External interrupt line 13
<b>EXTI_LINE_14</b>	External interrupt line 14
<b>EXTI_LINE_15</b>	External interrupt line 15
<b>EXTI_LINE_16</b>	External interrupt line 16 Connected to the PVD Output
<b>EXTI_LINE_17</b>	External interrupt line 17 Connected to the RTC Alarm event
<b>EXTI_LINE_18</b>	External interrupt line 18 Connected to the USB OTG FS Wakeup from suspend event
<b>EXTI_LINE_19</b>	External interrupt line 19 Connected to the Ethernet Wakeup event
<b>EXTI_LINE_20</b>	External interrupt line 20 Connected to the USB OTG HS (configured in FS) Wakeup event
<b>EXTI_LINE_21</b>	External interrupt line 21 Connected to the RTC Tamper and Time Stamp events
<b>EXTI_LINE_22</b>	External interrupt line 22 Connected to the RTC Wakeup event
	<b>EXTI Mode</b>
<b>EXTI_MODE_NONE</b>	
<b>EXTI_MODE_INTERRUPT</b>	
<b>EXTI_MODE_EVENT</b>	

*EXTI Trigger*

EXTI\_TRIGGER\_NONE

EXTI\_TRIGGER\_RISING

EXTI\_TRIGGER\_FALLING

EXTI\_TRIGGER\_RISING\_FALLING



## 26 HAL FLASH Generic Driver

### 26.1 FLASH Firmware driver registers structures

#### 26.1.1 FLASH\_ProcessTypeDef

*FLASH\_ProcessTypeDef* is defined in the `stm32f4xx_hal_flash.h`

##### Data Fields

- `__IO FLASH_ProcedureTypeDef ProcedureOnGoing`
- `__IO uint32_t NbSectorsToErase`
- `__IO uint8_t VoltageForErase`
- `__IO uint32_t Sector`
- `__IO uint32_t Bank`
- `__IO uint32_t Address`
- `HAL_LockTypeDef Lock`
- `__IO uint32_t ErrorCode`

##### Field Documentation

- `__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing`
- `__IO uint32_t FLASH_ProcessTypeDef::NbSectorsToErase`
- `__IO uint8_t FLASH_ProcessTypeDef::VoltageForErase`
- `__IO uint32_t FLASH_ProcessTypeDef::Sector`
- `__IO uint32_t FLASH_ProcessTypeDef::Bank`
- `__IO uint32_t FLASH_ProcessTypeDef::Address`
- `HAL_LockTypeDef FLASH_ProcessTypeDef::Lock`
- `__IO uint32_t FLASH_ProcessTypeDef::ErrorCode`

### 26.2 FLASH Firmware driver API description

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

#### 26.2.1 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
- Prefetch on I-Code
- 64 cache lines of 128 bits on I-Code
- 8 cache lines of 128 bits on D-Code

#### 26.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F4xx devices.

1. FLASH Memory IO Programming functions:
  - Lock and Unlock the FLASH interface using HAL\_FLASH\_Unlock() and HAL\_FLASH\_Lock() functions
  - Program functions: byte, half word, word and double word
  - There 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()
  - Wait for last FLASH operation according to its status
  - Get error flag status by calling HAL\_SetErrorCode()

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 interrupts
- Monitor the FLASH flags status

### 26.2.3 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\(\)\*](#)

### 26.2.4 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\(\)\*](#)

### 26.2.5 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\(\)\*](#)
- [\*FLASH\\_WaitForLastOperation\(\)\*](#)

### 26.2.6 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 byte, halfword, word or double word at a specified address.

**Parameters**

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program
- **Address:** specifies the address to be programmed.
- **Data:** specifies the data to be programmed

**Return values**

- **HAL\_StatusTypeDef:** 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 byte, halfword, word or double word 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 Type Program
- **Address:** specifies the address to be programmed.
- **Data:** specifies the data to be programmed

**Return values**

- **HAL:** Status

**HAL\_FLASH\_IRQHandler**
**Function name**
**void HAL\_FLASH\_IRQHandler (void )**
**Function description**

This function handles 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 Sectors Erase: Sector which has been erased (if 0xFFFFFFFFU, it means that all the selected sectors 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 Sectors Erase: Sector 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**

Locks 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 Control 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 Control 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 a combination of:
  - 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

### FLASH\_WaitForLastOperation

#### Function name

HAL\_StatusTypeDef FLASH\_WaitForLastOperation (uint32\_t Timeout)

#### Function description

Wait for a FLASH operation to complete.

#### Parameters

- **Timeout:** maximum flash operationtimeout

#### Return values

- **HAL:** Status

## 26.3 FLASH Firmware driver defines

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

### 26.3.1 FLASH

FLASH

*FLASH Error Code*

#### HAL\_FLASH\_ERROR\_NONE

No error

#### HAL\_FLASH\_ERROR\_RD

Read Protection error

#### HAL\_FLASH\_ERROR\_PGS

Programming Sequence error

#### HAL\_FLASH\_ERROR\_PGP

Programming Parallelism error

#### HAL\_FLASH\_ERROR\_PGA

Programming Alignment error

#### HAL\_FLASH\_ERROR\_WRP

Write protection error

#### HAL\_FLASH\_ERROR\_OPERATION

Operation Error

#### *FLASH Exported Macros*

#### \_\_HAL\_FLASH\_SET\_LATENCY

##### **Description:**

- Set the FLASH Latency.

##### **Parameters:**

- `__LATENCY__`: FLASH Latency The value of this parameter depend on device used within the same series

##### **Return value:**

- none

#### \_\_HAL\_FLASH\_GET\_LATENCY

##### **Description:**

- Get the FLASH Latency.

##### **Return value:**

- FLASH: Latency The value of this parameter depend on device used within the same series

#### \_\_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:**

- Resets 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:**

- Resets the FLASH data Cache.

**Return value:**

- None

**Notes:**

- This function must be used only when the data Cache is disabled.

#### `__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_ERR`: Error 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\_ERR: Error Interrupt

**Return value:**

- none

### **\_\_HAL\_FLASH\_GET\_FLAG**

**Description:**

- Get the specified FLASH flag status.

**Parameters:**

- **\_\_FLAG\_\_**: specifies the FLASH flags to check. 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\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGAERR: FLASH Programming Alignment error flag
  - FLASH\_FLAG\_PGPERR: FLASH Programming Parallelism error flag
  - FLASH\_FLAG\_PGSERR: FLASH Programming Sequence error flag
  - FLASH\_FLAG\_RDERR : FLASH Read Protection error flag (PCROP) (\*)
  - FLASH\_FLAG\_BSY : FLASH Busy flag (\*) FLASH\_FLAG\_RDERR is not available for STM32F405xx/407xx/415xx/417xx devices

**Return value:**

- The: new state of **\_\_FLAG\_\_** (SET or RESET).

### **\_\_HAL\_FLASH\_CLEAR\_FLAG**

**Description:**

- Clear the specified FLASH 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\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGAERR: FLASH Programming Alignment error flag
  - FLASH\_FLAG\_PGPERR: FLASH Programming Parallelism error flag
  - FLASH\_FLAG\_PGSERR: FLASH Programming Sequence error flag
  - FLASH\_FLAG\_RDERR : FLASH Read Protection error flag (PCROP) (\*) (\*) FLASH\_FLAG\_RDERR is not available for STM32F405xx/407xx/415xx/417xx devices

**Return value:**

- none

**FLASH Flag definition**

#### **FLASH\_FLAG\_EOP**

FLASH End of Operation flag

#### **FLASH\_FLAG\_OPERR**

FLASH operation Error flag



**FLASH\_FLAG\_WRPERR**

FLASH Write protected error flag

**FLASH\_FLAG\_PGAERR**

FLASH Programming Alignment error flag

**FLASH\_FLAG\_PGPERR**

FLASH Programming Parallelism error flag

**FLASH\_FLAG\_PGSERR**

FLASH Programming Sequence error flag

**FLASH\_FLAG\_RDERR**

Read Protection error flag (PCROP)

**FLASH\_FLAG\_BSY**

FLASH Busy flag

***FLASH Interrupt definition*****FLASH\_IT\_EOP**

End of FLASH Operation Interrupt source

**FLASH\_IT\_ERR**

Error Interrupt source

***FLASH Private macros to check input parameters*****IS\_FLASH\_TYPEPROGRAM*****FLASH Keys*****RDP\_KEY****FLASH\_KEY1****FLASH\_KEY2****FLASH\_OPT\_KEY1****FLASH\_OPT\_KEY2*****FLASH Latency*****FLASH\_LATENCY\_0**

FLASH Zero Latency cycle

**FLASH\_LATENCY\_1**

FLASH One Latency cycle

**FLASH\_LATENCY\_2**

FLASH Two Latency cycles

**FLASH\_LATENCY\_3**

FLASH Three Latency cycles

**FLASH\_LATENCY\_4**

FLASH Four Latency cycles

**FLASH\_LATENCY\_5**

FLASH Five Latency cycles

**FLASH\_LATENCY\_6**

FLASH Six Latency cycles

**FLASH\_LATENCY\_7**

FLASH Seven Latency cycles

**FLASH\_LATENCY\_8**

FLASH Eight Latency cycles

**FLASH\_LATENCY\_9**

FLASH Nine Latency cycles

**FLASH\_LATENCY\_10**

FLASH Ten Latency cycles

**FLASH\_LATENCY\_11**

FLASH Eleven Latency cycles

**FLASH\_LATENCY\_12**

FLASH Twelve Latency cycles

**FLASH\_LATENCY\_13**

FLASH Thirteen Latency cycles

**FLASH\_LATENCY\_14**

FLASH Fourteen Latency cycles

**FLASH\_LATENCY\_15**

FLASH Fifteen Latency cycles

***FLASH Program Parallelism*****FLASH\_PSIZE\_BYTE****FLASH\_PSIZE\_HALF\_WORD****FLASH\_PSIZE\_WORD****FLASH\_PSIZE\_DOUBLE\_WORD****CR\_PSIZE\_MASK*****FLASH Type Program*****FLASH\_TYPEPROGRAM\_BYTE**

Program byte (8-bit) at a specified address

**FLASH\_TYPEPROGRAM\_HALFWORD**

Program a half-word (16-bit) at a specified address

**FLASH\_TYPEPROGRAM\_WORD**

Program a word (32-bit) at a specified address

**FLASH\_TYPEPROGRAM\_DOUBLEWORD**

Program a double word (64-bit) at a specified address

## 27 HAL FLASH Extension Driver

### 27.1 FLASHEx Firmware driver registers structures

#### 27.1.1 FLASH\_EraseInitTypeDef

*FLASH\_EraseInitTypeDef* is defined in the `stm32f4xx_hal_flash_ex.h`

##### Data Fields

- *uint32\_t TypeErase*
- *uint32\_t Banks*
- *uint32\_t Sector*
- *uint32\_t NbSectors*
- *uint32\_t VoltageRange*

##### Field Documentation

- *uint32\_t FLASH\_EraseInitTypeDef::TypeErase*  
Mass erase or sector Erase. This parameter can be a value of [FLASHEx\\_Type\\_Erase](#)
- *uint32\_t FLASH\_EraseInitTypeDef::Banks*  
Select banks to erase when Mass erase is enabled. This parameter must be a value of [FLASHEx\\_Banks](#)
- *uint32\_t FLASH\_EraseInitTypeDef::Sector*  
Initial FLASH sector to erase when Mass erase is disabled This parameter must be a value of [FLASHEx\\_Sectors](#)
- *uint32\_t FLASH\_EraseInitTypeDef::NbSectors*  
Number of sectors to be erased. This parameter must be a value between 1 and (max number of sectors - value of Initial sector)
- *uint32\_t FLASH\_EraseInitTypeDef::VoltageRange*  
The device voltage range which defines the erase parallelism This parameter must be a value of [FLASHEx\\_Voltage\\_Range](#)

#### 27.1.2 FLASH\_OBProgramInitTypeDef

*FLASH\_OBProgramInitTypeDef* is defined in the `stm32f4xx_hal_flash_ex.h`

##### Data Fields

- *uint32\_t OptionType*
- *uint32\_t WRPState*
- *uint32\_t WRPSector*
- *uint32\_t Banks*
- *uint32\_t RDPLLevel*
- *uint32\_t BORLevel*
- *uint8\_t USERConfig*

##### Field Documentation

- *uint32\_t FLASH\_OBProgramInitTypeDef::OptionType*  
Option byte to be configured. This parameter can be a value of [FLASHEx\\_Option\\_Type](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPState*  
Write protection activation or deactivation. This parameter can be a value of [FLASHEx\\_WRP\\_State](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPSector*  
Specifies the sector(s) to be write protected. The value of this parameter depend on device used within the same series
- *uint32\_t FLASH\_OBProgramInitTypeDef::Banks*  
Select banks for WRP activation/deactivation of all sectors. This parameter must be a value of [FLASHEx\\_Banks](#)

- ***uint32\_t FLASH\_OBProgramInitTypeDef::RDPLLevel***  
Set the read protection level. This parameter can be a value of [FLASHEx\\_Option\\_Bytes\\_Read\\_Protection](#)
- ***uint32\_t FLASH\_OBProgramInitTypeDef::BORLevel***  
Set the BOR Level. This parameter can be a value of [FLASHEx\\_BOR\\_Reset\\_Level](#)
- ***uint8\_t FLASH\_OBProgramInitTypeDef::USERConfig***  
Program the FLASH User Option Byte: IWDG\_SW / RST\_STOP / RST\_STDBY.

### 27.1.3 FLASH\_AdvOBProgramInitTypeDef

*FLASH\_AdvOBProgramInitTypeDef* is defined in the `stm32f4xx_hal_flash_ex.h`

#### Data Fields

- ***uint32\_t OptionType***
- ***uint32\_t PCROPState***
- ***uint32\_t Banks***
- ***uint16\_t SectorsBank1***
- ***uint16\_t SectorsBank2***
- ***uint8\_t BootConfig***

#### Field Documentation

- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::OptionType***  
Option byte to be configured for extension. This parameter can be a value of [FLASHEx\\_Advanced\\_Option\\_Type](#)
- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::PCROPState***  
PCROP activation or deactivation. This parameter can be a value of [FLASHEx\\_PCROP\\_State](#)
- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::Banks***  
Select banks for PCROP activation/deactivation of all sectors. This parameter must be a value of [FLASHEx\\_Banks](#)
- ***uint16\_t FLASH\_AdvOBProgramInitTypeDef::SectorsBank1***  
Specifies the sector(s) set for PCROP for Bank1. This parameter can be a value of [FLASHEx\\_Option\\_Bytes\\_PC\\_ReadWrite\\_Protection](#)
- ***uint16\_t FLASH\_AdvOBProgramInitTypeDef::SectorsBank2***  
Specifies the sector(s) set for PCROP for Bank2. This parameter can be a value of [FLASHEx\\_Option\\_Bytes\\_PC\\_ReadWrite\\_Protection](#)
- ***uint8\_t FLASH\_AdvOBProgramInitTypeDef::BootConfig***  
Specifies Option bytes for boot config. This parameter can be a value of [FLASHEx\\_Dual\\_Boot](#)

## 27.2 FLASHEx Firmware driver API description

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

### 27.2.1 Flash Extension features

Comparing to other previous devices, the FLASH interface for STM32F427xx/437xx and STM32F429xx/439xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- PCROP protection for all banks

### 27.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F427xx/437xx, STM32F429xx/439xx, STM32F469xx/479xx and STM32F446xx 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 sector, erase all sectors
  - There are two modes of erase :
    - Polling Mode using HAL\_FLASHEx\_Erase()
    - Interrupt Mode using HAL\_FLASHEx\_Erase\_IT()
2. Option Bytes Programming functions: Use HAL\_FLASHEx\_OBProgram() to :
  - Set/Reset the write protection
  - Set the Read protection Level
  - Set the BOR level
  - Program the user Option Bytes
3. Advanced Option Bytes Programming functions: Use HAL\_FLASHEx\_AdvOBProgram() to :
  - Extended space (bank 2) erase function
  - Full FLASH space (2 Mo) erase (bank 1 and bank 2)
  - Dual Boot activation
  - Write protection configuration for bank 2
  - PCROP protection configuration and control for both banks

### 27.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extension FLASH programming operations.

This section contains the following APIs:

- [HAL\\_FLASHEx\\_Erase\(\)](#)
- [HAL\\_FLASHEx\\_Erase\\_IT\(\)](#)
- [HAL\\_FLASHEx\\_OBProgram\(\)](#)
- [HAL\\_FLASHEx\\_OBGetConfig\(\)](#)
- [HAL\\_FLASHEx\\_AdvOBProgram\(\)](#)
- [HAL\\_FLASHEx\\_AdvOBGetConfig\(\)](#)
- [HAL\\_FLASHEx\\_OB\\_SelectPCROP\(\)](#)
- [HAL\\_FLASHEx\\_OB\\_DeSelectPCROP\(\)](#)
- [HAL\\_FLASHEx\\_OB\\_GetBank2WRP\(\)](#)

### 27.2.4 Detailed description of functions

#### HAL\_FLASHEx\_Erase

##### Function name

**HAL\_StatusTypeDef HAL\_FLASHEx\_Erase (FLASH\_EraseInitTypeDef \* pEraseInit, uint32\_t \* SectorError)**

##### Function description

Perform a mass erase or erase the specified FLASH memory sectors.

##### Parameters

- **pEraseInit:** pointer to an FLASH\_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **SectorError:** pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFFU means that all the sectors 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 sectors 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

### HAL\_FLASHEx\_OBGetConfig

#### Function name

**void HAL\_FLASHEx\_OBGetConfig (FLASH\_OBProgramInitTypeDef \* pOBInit)**

#### Function description

Get the Option byte configuration.

#### Parameters

- **pOBInit:** pointer to an FLASH\_OBInitStruct structure that contains the configuration information for the programming.

#### Return values

- **None:**

### HAL\_FLASHEx\_AdvOBProgram

#### Function name

**HAL\_StatusTypeDef HAL\_FLASHEx\_AdvOBProgram (FLASH\_AdvOBProgramInitTypeDef \* pAdvOBInit)**

#### Function description

Program option bytes.

#### Parameters

- **pAdvOBInit:** pointer to an FLASH\_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.

**Return values**

- **HAL:** Status

**HAL\_FLASHEx\_AdvOBGetConfig**
**Function name**

```
void HAL_FLASHEx_AdvOBGetConfig (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)
```

**Function description**

Get the OBEX byte configuration.

**Parameters**

- **pAdvOBInit:** pointer to an FLASH\_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.

**Return values**

- **None:**

**HAL\_FLASHEx\_OB\_SelectPCROP**
**Function name**

```
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP (void )
```

**Function description**

Select the Protection Mode.

**Return values**

- **HAL:** Status

**Notes**

- After PCROP activated Option Byte modification NOT POSSIBLE! excepted Global Read Out Protection modification (from level1 to level0)
- Once SPRMOD bit is active unprotection of a protected sector is not possible
- Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag
- This function can be used only for STM32F42xxx/STM32F43xxx/STM32F401xx/STM32F411xx/STM32F446xx/ STM32F469xx/STM32F479xx/STM32F412xx/STM32F413xx devices.

**HAL\_FLASHEx\_OB\_DeSelectPCROP**
**Function name**

```
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP (void )
```

**Function description**

Deselect the Protection Mode.

**Return values**

- **HAL:** Status

**Notes**

- After PCROP activated Option Byte modification NOT POSSIBLE! excepted Global Read Out Protection modification (from level1 to level0)
- Once SPRMOD bit is active unprotection of a protected sector is not possible
- Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag
- This function can be used only for STM32F42xxx/STM32F43xxx/STM32F401xx/STM32F411xx/STM32F446xx/ STM32F469xx/STM32F479xx/STM32F412xx/STM32F413xx devices.

### HAL\_FLASHEx\_OB\_GetBank2WRP

#### Function name

**uint16\_t HAL\_FLASHEx\_OB\_GetBank2WRP (void )**

#### Function description

Returns the FLASH Write Protection Option Bytes value for Bank 2.

#### Return values

- **The:** FLASH Write Protection Option Bytes value

#### Notes

- This function can be used only for STM32F42xxx/STM32F43xxx/STM32F469xx/STM32F479xx devices.

### FLASH\_Erase\_Sector

#### Function name

**void FLASH\_Erase\_Sector (uint32\_t Sector, uint8\_t VoltageRange)**

#### Function description

Erase the specified FLASH memory sector.

#### Parameters

- **Sector:** FLASH sector to erase The value of this parameter depend on device used within the same series
- **VoltageRange:** The device voltage range which defines the erase parallelism. This parameter can be one of the following values:
  - FLASH\_VOLTAGE\_RANGE\_1: when the device voltage range is 1.8V to 2.1V, the operation will be done by byte (8-bit)
  - FLASH\_VOLTAGE\_RANGE\_2: when the device voltage range is 2.1V to 2.7V, the operation will be done by half word (16-bit)
  - FLASH\_VOLTAGE\_RANGE\_3: when the device voltage range is 2.7V to 3.6V, the operation will be done by word (32-bit)
  - FLASH\_VOLTAGE\_RANGE\_4: when the device voltage range is 2.7V to 3.6V + External Vpp, the operation will be done by double word (64-bit)

#### Return values

- **None:**

### FLASH\_FlushCaches

#### Function name

**void FLASH\_FlushCaches (void )**

#### Function description

Flush the instruction and data caches.

#### Return values

- **None:**

## 27.3 FLASHEx Firmware driver defines

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

### 27.3.1

#### FLASHEx

FLASHEx

*FLASH Advanced Option Type*



**OPTIONBYTE\_PFCROP**

PFCROP option byte configuration

**OPTIONBYTE\_BOOTCONFIG**

BOOTConfig option byte configuration

**FLASH Banks****FLASH\_BANK\_1**

Bank 1

**FLASH\_BANK\_2**

Bank 2

**FLASH\_BANK\_BOTH**

Bank1 and Bank2

**FLASH BOR Reset Level****OB\_BOR\_LEVEL3**

Supply voltage ranges from 2.70 to 3.60 V

**OB\_BOR\_LEVEL2**

Supply voltage ranges from 2.40 to 2.70 V

**OB\_BOR\_LEVEL1**

Supply voltage ranges from 2.10 to 2.40 V

**OB\_BOR\_OFF**

Supply voltage ranges from 1.62 to 2.10 V

**FLASH Dual Boot****OB\_DUAL\_BOOT\_ENABLE**

Dual Bank Boot Enable

**OB\_DUAL\_BOOT\_DISABLE**

Dual Bank Boot Disable, always boot on User Flash

**FLASH Private macros to check input parameters****IS\_FLASH\_TYPEERASE****IS\_VOLTAGERANGE****IS\_WRPSTATE****IS\_OPTIONBYTE****IS\_OB\_RDP\_LEVEL****IS\_OB\_IWDG\_SOURCE****IS\_OB\_STOP\_SOURCE****IS\_OB\_STDBY\_SOURCE****IS\_OB\_BOR\_LEVEL****IS\_PFCROPSTATE**

IS\_OBEX

IS\_FLASH\_LATENCY

IS\_FLASH\_BANK

IS\_FLASH\_SECTOR

IS\_FLASH\_ADDRESS

IS\_FLASH\_NBSECTORS

IS\_OB\_WRP\_SECTOR

IS\_OB\_PCROP

IS\_OB\_BOOT

IS\_OB\_PCROP\_SELECT

***FLASH Mass Erase bit***

FLASH\_MER\_BIT

2 MER bits here to clear

***FLASH Option Bytes IWatchdog***

OB\_IWDG\_SW

Software IWDG selected

OB\_IWDG\_HW

Hardware IWDG selected

***FLASH Option Bytes nRST\_STDBY***

OB\_STDBY\_NO\_RST

No reset generated when entering in STANDBY

OB\_STDBY\_RST

Reset generated when entering in STANDBY

***FLASH Option Bytes nRST\_STOP***

OB\_STOP\_NO\_RST

No reset generated when entering in STOP

OB\_STOP\_RST

Reset generated when entering in STOP

***FLASH Option Bytes PC ReadWrite Protection***

OB\_PCROP\_SECTOR\_0

PC Read/Write protection of Sector0

OB\_PCROP\_SECTOR\_1

PC Read/Write protection of Sector1

OB\_PCROP\_SECTOR\_2

PC Read/Write protection of Sector2

**OB\_PCROP\_SECTOR\_3**

PC Read/Write protection of Sector3

**OB\_PCROP\_SECTOR\_4**

PC Read/Write protection of Sector4

**OB\_PCROP\_SECTOR\_5**

PC Read/Write protection of Sector5

**OB\_PCROP\_SECTOR\_6**

PC Read/Write protection of Sector6

**OB\_PCROP\_SECTOR\_7**

PC Read/Write protection of Sector7

**OB\_PCROP\_SECTOR\_8**

PC Read/Write protection of Sector8

**OB\_PCROP\_SECTOR\_9**

PC Read/Write protection of Sector9

**OB\_PCROP\_SECTOR\_10**

PC Read/Write protection of Sector10

**OB\_PCROP\_SECTOR\_11**

PC Read/Write protection of Sector11

**OB\_PCROP\_SECTOR\_12**

PC Read/Write protection of Sector12

**OB\_PCROP\_SECTOR\_13**

PC Read/Write protection of Sector13

**OB\_PCROP\_SECTOR\_14**

PC Read/Write protection of Sector14

**OB\_PCROP\_SECTOR\_15**

PC Read/Write protection of Sector15

**OB\_PCROP\_SECTOR\_16**

PC Read/Write protection of Sector16

**OB\_PCROP\_SECTOR\_17**

PC Read/Write protection of Sector17

**OB\_PCROP\_SECTOR\_18**

PC Read/Write protection of Sector18

**OB\_PCROP\_SECTOR\_19**

PC Read/Write protection of Sector19

**OB\_PCROP\_SECTOR\_20**

PC Read/Write protection of Sector20

**OB\_PCROP\_SECTOR\_21**

PC Read/Write protection of Sector21

**OB\_PCROP\_SECTOR\_22**

PC Read/Write protection of Sector22

**OB\_PCROP\_SECTOR\_23**

PC Read/Write protection of Sector23

**OB\_PCROP\_SECTOR\_All**

PC Read/Write protection of all Sectors

**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 is no more possible to go back to level 1 or 0

**FLASH Option Bytes Write Protection****OB\_WRP\_SECTOR\_0**

Write protection of Sector0

**OB\_WRP\_SECTOR\_1**

Write protection of Sector1

**OB\_WRP\_SECTOR\_2**

Write protection of Sector2

**OB\_WRP\_SECTOR\_3**

Write protection of Sector3

**OB\_WRP\_SECTOR\_4**

Write protection of Sector4

**OB\_WRP\_SECTOR\_5**

Write protection of Sector5

**OB\_WRP\_SECTOR\_6**

Write protection of Sector6

**OB\_WRP\_SECTOR\_7**

Write protection of Sector7

**OB\_WRP\_SECTOR\_8**

Write protection of Sector8

**OB\_WRP\_SECTOR\_9**

Write protection of Sector9

**OB\_WRP\_SECTOR\_10**

Write protection of Sector10

**OB\_WRP\_SECTOR\_11**

Write protection of Sector11

**OB\_WRP\_SECTOR\_12**

Write protection of Sector12

**OB\_WRP\_SECTOR\_13**

Write protection of Sector13

**OB\_WRP\_SECTOR\_14**

Write protection of Sector14

**OB\_WRP\_SECTOR\_15**

Write protection of Sector15

**OB\_WRP\_SECTOR\_16**

Write protection of Sector16

**OB\_WRP\_SECTOR\_17**

Write protection of Sector17

**OB\_WRP\_SECTOR\_18**

Write protection of Sector18

**OB\_WRP\_SECTOR\_19**

Write protection of Sector19

**OB\_WRP\_SECTOR\_20**

Write protection of Sector20

**OB\_WRP\_SECTOR\_21**

Write protection of Sector21

**OB\_WRP\_SECTOR\_22**

Write protection of Sector22

**OB\_WRP\_SECTOR\_23**

Write protection of Sector23

**OB\_WRP\_SECTOR\_All**

Write protection of all Sectors

***FLASH Option Type*****OPTIONBYTE\_WRP**

WRP option byte configuration

**OPTIONBYTE\_RDP**

RDP option byte configuration

**OPTIONBYTE\_USER**

USER option byte configuration

**OPTIONBYTE\_BOR**

BOR option byte configuration

***FLASH PCROP State*****OB\_PCROP\_STATE\_DISABLE**

Disable PCROP

**OB\_PCROP\_STATE\_ENABLE**

Enable PCROP

***FLASH Sectors***

**FLASH\_SECTOR\_0**

Sector Number 0

**FLASH\_SECTOR\_1**

Sector Number 1

**FLASH\_SECTOR\_2**

Sector Number 2

**FLASH\_SECTOR\_3**

Sector Number 3

**FLASH\_SECTOR\_4**

Sector Number 4

**FLASH\_SECTOR\_5**

Sector Number 5

**FLASH\_SECTOR\_6**

Sector Number 6

**FLASH\_SECTOR\_7**

Sector Number 7

**FLASH\_SECTOR\_8**

Sector Number 8

**FLASH\_SECTOR\_9**

Sector Number 9

**FLASH\_SECTOR\_10**

Sector Number 10

**FLASH\_SECTOR\_11**

Sector Number 11

**FLASH\_SECTOR\_12**

Sector Number 12

**FLASH\_SECTOR\_13**

Sector Number 13

**FLASH\_SECTOR\_14**

Sector Number 14

**FLASH\_SECTOR\_15**

Sector Number 15

**FLASH\_SECTOR\_16**

Sector Number 16

**FLASH\_SECTOR\_17**

Sector Number 17

**FLASH\_SECTOR\_18**

Sector Number 18

**FLASH\_SECTOR\_19**

Sector Number 19

**FLASH\_SECTOR\_20**

Sector Number 20

**FLASH\_SECTOR\_21**

Sector Number 21

**FLASH\_SECTOR\_22**

Sector Number 22

**FLASH\_SECTOR\_23**

Sector Number 23

**FLASH Selection Protection Mode****OB\_PCROP\_DESELECTED**

Disabled PcROP, nWPRI bits used for Write Protection on sector i

**OB\_PCROP\_SELECTED**

Enable PcROP, nWPRI bits used for PCRoP Protection on sector i

**FLASH Type Erase****FLASH\_TYPEERASE\_SECTORS**

Sectors erase only

**FLASH\_TYPEERASE\_MASSERASE**

Flash Mass erase activation

**FLASH Voltage Range****FLASH\_VOLTAGE\_RANGE\_1**

Device operating range: 1.8V to 2.1V

**FLASH\_VOLTAGE\_RANGE\_2**

Device operating range: 2.1V to 2.7V

**FLASH\_VOLTAGE\_RANGE\_3**

Device operating range: 2.7V to 3.6V

**FLASH\_VOLTAGE\_RANGE\_4**

Device operating range: 2.7V to 3.6V + External Vpp

**FLASH WRP State****OB\_WRPSTATE\_DISABLE**

Disable the write protection of the desired bank 1 sectors

**OB\_WRPSTATE\_ENABLE**

Enable the write protection of the desired bank 1 sectors

## 28 HAL FLASH\_\_RAMFUNC Generic Driver

### 28.1 FLASH\_\_RAMFUNC Firmware driver API description

The following section lists the various functions of the FLASH\_\_RAMFUNC library.

#### 28.1.1 APIs executed from Internal RAM

##### ARM Compiler

RAM functions are defined using the toolchain options. Functions that are to be 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")))`".

#### 28.1.2 ramfunc functions

This subsection provides a set of functions that should be executed from RAM transfers.

This section contains the following APIs:

- [\*HAL\\_FLASHEx\\_StopFlashInterfaceClk\(\)\*](#)
- [\*HAL\\_FLASHEx\\_StartFlashInterfaceClk\(\)\*](#)
- [\*HAL\\_FLASHEx\\_EnableFlashSleepMode\(\)\*](#)
- [\*HAL\\_FLASHEx\\_DisableFlashSleepMode\(\)\*](#)

#### 28.1.3 Detailed description of functions

##### HAL\_FLASHEx\_StopFlashInterfaceClk

###### Function name

`__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_StopFlashInterfaceClk (void )`

###### Function description

Stop the flash interface while System Run.

###### Return values

- **HAL:** status

###### Notes

- This mode is only available for STM32F41xxx/STM32F446xx devices.
- This mode couldn't be set while executing with the flash itself. It should be done with specific routine executed from RAM.

##### HAL\_FLASHEx\_StartFlashInterfaceClk

###### Function name

`__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_StartFlashInterfaceClk (void )`

###### Function description

Start the flash interface while System Run.



**Return values**

- **HAL:** status

**Notes**

- This mode is only available for STM32F411xx/STM32F446xx devices.
- This mode couldn't be set while executing with the flash itself. It should be done with specific routine executed from RAM.

**HAL\_FLASHEx\_EnableFlashSleepMode**
**Function name**

`__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_EnableFlashSleepMode (void )`

**Function description**

Enable the flash sleep while System Run.

**Return values**

- **HAL:** status

**Notes**

- This mode is only available for STM32F41xxx/STM32F446xx devices.
- This mode could n't be set while executing with the flash itself. It should be done with specific routine executed from RAM.

**HAL\_FLASHEx\_DisableFlashSleepMode**
**Function name**

`__RAM_FUNC HAL_StatusTypeDef HAL_FLASHEx_DisableFlashSleepMode (void )`

**Function description**

Disable the flash sleep while System Run.

**Return values**

- **HAL:** status

**Notes**

- This mode is only available for STM32F41xxx/STM32F446xx devices.
- This mode couldn't be set while executing with the flash itself. It should be done with specific routine executed from RAM.

## 29 HAL FMPI2C Generic Driver

### 29.1 FMPI2C Firmware driver registers structures

#### 29.1.1 FMPI2C\_InitTypeDef

*FMPI2C\_InitTypeDef* is defined in the `stm32f4xx_hal_fmapi2c.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* *FMPI2C\_InitTypeDef::Timing*  
Specifies the `FMPI2C_TIMINGR` register value. This parameter calculated by referring to FMPI2C initialization section in Reference manual
- *uint32\_t* *FMPI2C\_InitTypeDef::OwnAddress1*  
Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- *uint32\_t* *FMPI2C\_InitTypeDef::AddressingMode*  
Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [FMPI2C\\_ADDRESSING\\_MODE](#)
- *uint32\_t* *FMPI2C\_InitTypeDef::DualAddressMode*  
Specifies if dual addressing mode is selected. This parameter can be a value of [FMPI2C\\_DUAL\\_ADDRESSING\\_MODE](#)
- *uint32\_t* *FMPI2C\_InitTypeDef::OwnAddress2*  
Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- *uint32\_t* *FMPI2C\_InitTypeDef::OwnAddress2Masks*  
Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of [FMPI2C\\_OWN\\_ADDRESS2\\_MASKS](#)
- *uint32\_t* *FMPI2C\_InitTypeDef::GeneralCallMode*  
Specifies if general call mode is selected. This parameter can be a value of [FMPI2C\\_GENERAL\\_CALL\\_ADDRESSING\\_MODE](#)
- *uint32\_t* *FMPI2C\_InitTypeDef::NoStretchMode*  
Specifies if nostretch mode is selected. This parameter can be a value of [FMPI2C\\_NOSTRETCH\\_MODE](#)

#### 29.1.2 \_\_FMPI2C\_HandleTypeDef

*\_\_FMPI2C\_HandleTypeDef* is defined in the `stm32f4xx_hal_fmapi2c.h`

##### Data Fields

- *FMPI2C\_TypeDef \* Instance*
- *FMPI2C\_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 \* hdmatrix***
- ***DMA\_HandleTypeDef \* hdmatrix***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_FMPI2C\_StateTypeDef State***
- ***\_\_IO HAL\_FMPI2C\_ModeTypeDef Mode***
- ***\_\_IO uint32\_t ErrorCode***
- ***\_\_IO uint32\_t AddrEventCount***

#### Field Documentation

- ***FMPI2C\_TypeDef\* \_\_FMPI2C\_HandleTypeDef::Instance***  
FMPI2C registers base address
- ***FMPI2C\_InitTypeDef \_\_FMPI2C\_HandleTypeDef::Init***  
FMPI2C communication parameters
- ***uint8\_t\* \_\_FMPI2C\_HandleTypeDef::pBuffPtr***  
Pointer to FMPI2C transfer buffer
- ***uint16\_t \_\_FMPI2C\_HandleTypeDef::XferSize***  
FMPI2C transfer size
- ***\_\_IO uint16\_t \_\_FMPI2C\_HandleTypeDef::XferCount***  
FMPI2C transfer counter
- ***\_\_IO uint32\_t \_\_FMPI2C\_HandleTypeDef::XferOptions***  
FMPI2C sequantial transfer options, this parameter can be a value of [FMPI2C\\_XFEROPTIONS](#)
- ***\_\_IO uint32\_t \_\_FMPI2C\_HandleTypeDef::PreviousState***  
FMPI2C communication Previous state
- ***HAL\_StatusTypeDef(\* \_\_FMPI2C\_HandleTypeDef::XferISR)(struct \_\_FMPI2C\_HandleTypeDef \*hfmipi2c, uint32\_t ITFlags, uint32\_t ITSources)***  
FMPI2C transfer IRQ handler function pointer
- ***DMA\_HandleTypeDef\* \_\_FMPI2C\_HandleTypeDef::hdmatrix***  
FMPI2C Tx DMA handle parameters
- ***DMA\_HandleTypeDef\* \_\_FMPI2C\_HandleTypeDef::hdmatrix***  
FMPI2C Rx DMA handle parameters
- ***HAL\_LockTypeDef \_\_FMPI2C\_HandleTypeDef::Lock***  
FMPI2C locking object
- ***\_\_IO HAL\_FMPI2C\_StateTypeDef \_\_FMPI2C\_HandleTypeDef::State***  
FMPI2C communication state
- ***\_\_IO HAL\_FMPI2C\_ModeTypeDef \_\_FMPI2C\_HandleTypeDef::Mode***  
FMPI2C communication mode
- ***\_\_IO uint32\_t \_\_FMPI2C\_HandleTypeDef::ErrorCode***  
FMPI2C Error code
- ***\_\_IO uint32\_t \_\_FMPI2C\_HandleTypeDef::AddrEventCount***  
FMPI2C Address Event counter

## 29.2 FMPI2C Firmware driver API description

The following section lists the various functions of the FMPI2C library.

### 29.2.1 How to use this driver

The FMPI2C HAL driver can be used as follows:

1. Declare a `FMPI2C_HandleTypeDef` handle structure, for example: `FMPI2C_HandleTypeDef hfmipi2c;`

2. Initialize the FMPI2C low level resources by implementing the @ref HAL\_FMPI2C\_MspInit() API:
  - a. Enable the FMPI2Cx interface clock
  - b. FMPI2C pins configuration
    - Enable the clock for the FMPI2C GPIOs
    - Configure FMPI2C pins as alternate function open-drain
  - c. NVIC configuration if you need to use interrupt process
    - Configure the FMPI2Cx interrupt priority
    - Enable the NVIC FMPI2C IRQ Channel
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive stream
    - Enable the DMAx interface clock using
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx stream
    - Associate the initialized DMA handle to the hfmipi2c DMA Tx or Rx handle
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx stream
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 hfmipi2c Init structure.
4. Initialize the FMPI2C registers by calling the @ref HAL\_FMPI2C\_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized @ref HAL\_FMPI2C\_MspInit(&hfmipi2c) API.
5. To check if target device is ready for communication, use the function @ref HAL\_FMPI2C\_IsDeviceReady()
6. For FMPI2C 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\_FMPI2C\_Master\_Transmit()
- Receive in master mode an amount of data in blocking mode using @ref HAL\_FMPI2C\_Master\_Receive()
- Transmit in slave mode an amount of data in blocking mode using @ref HAL\_FMPI2C\_Slave\_Transmit()
- Receive in slave mode an amount of data in blocking mode using @ref HAL\_FMPI2C\_Slave\_Receive()

#### **Polling mode IO MEM operation**

- Write an amount of data in blocking mode to a specific memory address using @ref HAL\_FMPI2C\_Mem\_Write()
- Read an amount of data in blocking mode from a specific memory address using @ref HAL\_FMPI2C\_Mem\_Read()

#### **Interrupt mode IO operation**

- Transmit in master mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Master\_Transmit\_IT()
- At transmission end of transfer, @ref HAL\_FMPI2C\_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Master\_Receive\_IT()
- At reception end of transfer, @ref HAL\_FMPI2C\_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Slave\_Transmit\_IT()
- At transmission end of transfer, @ref HAL\_FMPI2C\_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Slave\_Receive\_IT()
- At reception end of transfer, @ref HAL\_FMPI2C\_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_SlaveRxCpltCallback()

- In case of transfer Error, @ref HAL\_FMPI2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_ErrorCallback()
- Abort a master FMPI2C process communication with Interrupt using @ref HAL\_FMPI2C\_Master\_Abort\_IT()
- End of abort process, @ref HAL\_FMPI2C\_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_AbortCpltCallback()
- Discard a slave FMPI2C process communication using @ref \_\_HAL\_FMPI2C\_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 FMPI2C\_XFEROPTIONS and are listed below:
  - FMPI2C\_FIRST\_AND\_LAST\_FRAME: No sequential usage, functional is same as associated interfaces in no sequential mode
  - FMPI2C\_FIRST\_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - FMPI2C\_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\_FMPI2C\_Master\_Seq\_Transmit\_IT() then @ref HAL\_FMPI2C\_Master\_Seq\_Transmit\_IT() or @ref HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA() then @ref HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA())
  - FMPI2C\_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
  - FMPI2C\_LAST\_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 with a final stop condition in both cases
  - FMPI2C\_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 FMPI2C\_FIRST\_AND\_NEXT\_FRAME). Usage can, transfer several bytes one by one using HAL\_FMPI2C\_Master\_Seq\_Transmit\_IT(option FMPI2C\_FIRST\_AND\_NEXT\_FRAME then FMPI2C\_NEXT\_FRAME) or HAL\_FMPI2C\_Master\_Seq\_Receive\_IT(option FMPI2C\_FIRST\_AND\_NEXT\_FRAME then FMPI2C\_NEXT\_FRAME) or HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA(option FMPI2C\_FIRST\_AND\_NEXT\_FRAME then FMPI2C\_NEXT\_FRAME) or HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA(option FMPI2C\_FIRST\_AND\_NEXT\_FRAME then FMPI2C\_NEXT\_FRAME). Then usage of this option FMPI2C\_LAST\_FRAME\_NO\_STOP at the last Transmit or Receive sequence permit to call the opposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - FMPI2C\_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\_FMPI2C\_Master\_Seq\_Transmit\_IT(option FMPI2C\_FIRST\_FRAME then FMPI2C\_OTHER\_FRAME) or HAL\_FMPI2C\_Master\_Seq\_Receive\_IT(option FMPI2C\_FIRST\_FRAME then FMPI2C\_OTHER\_FRAME) or HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA(option FMPI2C\_FIRST\_FRAME then FMPI2C\_OTHER\_FRAME) or HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA(option FMPI2C\_FIRST\_FRAME then FMPI2C\_OTHER\_FRAME). Then usage of this option FMPI2C\_OTHER\_AND\_LAST\_FRAME at the last frame to help automatic generation of STOP condition.

- Differents sequential FMPI2C interfaces are listed below:
  - Sequential transmit in master FMPI2C mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Master\_Seq\_Transmit\_IT() or using @ref HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA()
    - At transmission end of current frame transfer, @ref HAL\_FMPI2C\_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MasterTxCpltCallback()
  - Sequential receive in master FMPI2C mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Master\_Seq\_Receive\_IT() or using @ref HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA()
    - At reception end of current frame transfer, @ref HAL\_FMPI2C\_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MasterRxCpltCallback()
  - Abort a master IT or DMA FMPI2C process communication with Interrupt using @ref HAL\_FMPI2C\_Master\_Abort\_IT()
    - End of abort process, @ref HAL\_FMPI2C\_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave FMPI2C mode using @ref HAL\_FMPI2C\_EnableListen\_IT() @ref HAL\_FMPI2C\_DisableListen\_IT()
    - When address slave FMPI2C match, @ref HAL\_FMPI2C\_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\_FMPI2C\_ListenCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_ListenCpltCallback()
  - Sequential transmit in slave FMPI2C mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Slave\_Seq\_Transmit\_IT() or using @ref HAL\_FMPI2C\_Slave\_Seq\_Transmit\_DMA()
    - At transmission end of current frame transfer, @ref HAL\_FMPI2C\_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_SlaveTxCpltCallback()
  - Sequential receive in slave FMPI2C mode an amount of data in non-blocking mode using @ref HAL\_FMPI2C\_Slave\_Seq\_Receive\_IT() or using @ref HAL\_FMPI2C\_Slave\_Seq\_Receive\_DMA()
    - At reception end of current frame transfer, @ref HAL\_FMPI2C\_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_SlaveRxCpltCallback()
  - In case of transfer Error, @ref HAL\_FMPI2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_ErrorCallback()
  - Discard a slave FMPI2C process communication using @ref \_\_HAL\_FMPI2C\_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\_FMPI2C\_Mem\_Write\_IT()
- At Memory end of write transfer, @ref HAL\_FMPI2C\_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using @ref HAL\_FMPI2C\_Mem\_Read\_IT()
- At Memory end of read transfer, @ref HAL\_FMPI2C\_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MemRxCpltCallback()
- In case of transfer Error, @ref HAL\_FMPI2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_ErrorCallback()

#### **DMA mode IO operation**

- Transmit in master mode an amount of data in non-blocking mode (DMA) using @ref HAL\_FMPI2C\_Master\_Transmit\_DMA()
- At transmission end of transfer, @ref HAL\_FMPI2C\_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode (DMA) using @ref HAL\_FMPI2C\_Master\_Receive\_DMA()

- At reception end of transfer, @ref HAL\_FMPI2C\_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using @ref HAL\_FMPI2C\_Slave\_Transmit\_DMA()
- At transmission end of transfer, @ref HAL\_FMPI2C\_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using @ref HAL\_FMPI2C\_Slave\_Receive\_DMA()
- At reception end of transfer, @ref HAL\_FMPI2C\_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_SlaveRxCpltCallback()
- In case of transfer Error, @ref HAL\_FMPI2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_ErrorCallback()
- Abort a master FMPI2C process communication with Interrupt using @ref HAL\_FMPI2C\_Master\_Abort\_IT()
- End of abort process, @ref HAL\_FMPI2C\_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_AbortCpltCallback()
- Discard a slave FMPI2C process communication using @ref \_\_HAL\_FMPI2C\_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\_FMPI2C\_Mem\_Write\_DMA()
- At Memory end of write transfer, @ref HAL\_FMPI2C\_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using @ref HAL\_FMPI2C\_Mem\_Read\_DMA()
- At Memory end of read transfer, @ref HAL\_FMPI2C\_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_MemRxCpltCallback()
- In case of transfer Error, @ref HAL\_FMPI2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL\_FMPI2C\_ErrorCallback()

#### **FMPI2C HAL driver macros list**

Below the list of most used macros in FMPI2C HAL driver.

- @ref \_\_HAL\_FMPI2C\_ENABLE: Enable the FMPI2C peripheral
- @ref \_\_HAL\_FMPI2C\_DISABLE: Disable the FMPI2C peripheral
- @ref \_\_HAL\_FMPI2C\_GENERATE\_NACK: Generate a Non-Acknowledge FMPI2C peripheral in Slave mode
- @ref \_\_HAL\_FMPI2C\_GET\_FLAG: Check whether the specified FMPI2C flag is set or not
- @ref \_\_HAL\_FMPI2C\_CLEAR\_FLAG: Clear the specified FMPI2C pending flag
- @ref \_\_HAL\_FMPI2C\_ENABLE\_IT: Enable the specified FMPI2C interrupt
- @ref \_\_HAL\_FMPI2C\_DISABLE\_IT: Disable the specified FMPI2C interrupt

#### **Callback registration**

The compilation flag `USE_HAL_FMPI2C_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL\_FMPI2C\_RegisterCallback() or @ref HAL\_FMPI2C\_RegisterAddrCallback() to register an interrupt callback.

Function @ref HAL\_FMPI2C\_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.
- MspDelInitCallback : callback for Msp Delnit. 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\_FMPI2C\_RegisterAddrCallback().

Use function @ref HAL\_FMPI2C\_UnRegisterCallback to reset a callback to the default weak function. @ref HAL\_FMPI2C\_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.
- MspDelInitCallback : callback for Msp Delnit.

For callback AddrCallback use dedicated register callbacks : @ref HAL\_FMPI2C\_UnRegisterAddrCallback().

By default, after the @ref HAL\_FMPI2C\_Init() and when the state is @ref HAL\_FMPI2C\_STATE\_RESET all callbacks are set to the corresponding weak functions: examples @ref HAL\_FMPI2C\_MasterTxCpltCallback(), @ref HAL\_FMPI2C\_MasterRxCpltCallback(). Exception done for MspInit and MspDelnit functions that are reset to the legacy weak functions in the @ref HAL\_FMPI2C\_Init()/ @ref HAL\_FMPI2C\_Delnit() only when these callbacks are null (not registered beforehand). If MspInit or MspDelnit are not null, the @ref HAL\_FMPI2C\_Init()/ @ref HAL\_FMPI2C\_Delnit() keep and use the user MspInit/MspDelnit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL\_FMPI2C\_STATE\_READY state only. Exception done MspInit/MspDelnit functions that can be registered/unregistered in @ref HAL\_FMPI2C\_STATE\_READY or @ref HAL\_FMPI2C\_STATE\_RESET state, thus registered (user) MspInit/Delnit callbacks can be used during the Init/Delnit. Then, the user first registers the MspInit/MspDelnit user callbacks using @ref HAL\_FMPI2C\_RegisterCallback() before calling @ref HAL\_FMPI2C\_Delnit() or @ref HAL\_FMPI2C\_Init() function.

When the compilation flag USE\_HAL\_FMPI2C\_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 FMPI2C 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 FMPI2Cx peripheral:

- User must Implement HAL\_FMPI2C\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_FMPI2C\_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\_FMPI2C\_Delnit() to restore the default configuration of the selected FMPI2Cx peripheral.



This section contains the following APIs:

- ***HAL\_FMPI2C\_Init()***
- ***HAL\_FMPI2C\_DeInit()***
- ***HAL\_FMPI2C\_Msplnit()***
- ***HAL\_FMPI2C\_MspDeInit()***

### 29.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the FMPI2C 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 FMPI2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
  - HAL\_FMPI2C\_Master\_Transmit()
  - HAL\_FMPI2C\_Master\_Receive()
  - HAL\_FMPI2C\_Slave\_Transmit()
  - HAL\_FMPI2C\_Slave\_Receive()
  - HAL\_FMPI2C\_Mem\_Write()
  - HAL\_FMPI2C\_Mem\_Read()
  - HAL\_FMPI2C\_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
  - HAL\_FMPI2C\_Master\_Transmit\_IT()
  - HAL\_FMPI2C\_Master\_Receive\_IT()
  - HAL\_FMPI2C\_Slave\_Transmit\_IT()
  - HAL\_FMPI2C\_Slave\_Receive\_IT()
  - HAL\_FMPI2C\_Mem\_Write\_IT()
  - HAL\_FMPI2C\_Mem\_Read\_IT()
  - HAL\_FMPI2C\_Master\_Seq\_Transmit\_IT()
  - HAL\_FMPI2C\_Master\_Seq\_Receive\_IT()
  - HAL\_FMPI2C\_Slave\_Seq\_Transmit\_IT()
  - HAL\_FMPI2C\_Slave\_Seq\_Receive\_IT()
  - HAL\_FMPI2C\_EnableListen\_IT()
  - HAL\_FMPI2C\_DisableListen\_IT()
  - HAL\_FMPI2C\_Master\_Abort\_IT()
4. No-Blocking mode functions with DMA are :
  - HAL\_FMPI2C\_Master\_Transmit\_DMA()
  - HAL\_FMPI2C\_Master\_Receive\_DMA()
  - HAL\_FMPI2C\_Slave\_Transmit\_DMA()
  - HAL\_FMPI2C\_Slave\_Receive\_DMA()
  - HAL\_FMPI2C\_Mem\_Write\_DMA()
  - HAL\_FMPI2C\_Mem\_Read\_DMA()
  - HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA()
  - HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA()
  - HAL\_FMPI2C\_Slave\_Seq\_Transmit\_DMA()
  - HAL\_FMPI2C\_Slave\_Seq\_Receive\_DMA()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:

- HAL\_FMPI2C\_MasterTxCpltCallback()
- HAL\_FMPI2C\_MasterRxCpltCallback()
- HAL\_FMPI2C\_SlaveTxCpltCallback()
- HAL\_FMPI2C\_SlaveRxCpltCallback()
- HAL\_FMPI2C\_MemTxCpltCallback()
- HAL\_FMPI2C\_MemRxCpltCallback()
- HAL\_FMPI2C\_AddrCallback()
- HAL\_FMPI2C\_ListenCpltCallback()
- HAL\_FMPI2C\_ErrorCallback()
- HAL\_FMPI2C\_AbortCpltCallback()

This section contains the following APIs:

- *HAL\_FMPI2C\_Master\_Transmit()*
- *HAL\_FMPI2C\_Master\_Receive()*
- *HAL\_FMPI2C\_Slave\_Transmit()*
- *HAL\_FMPI2C\_Slave\_Receive()*
- *HAL\_FMPI2C\_Master\_Transmit\_IT()*
- *HAL\_FMPI2C\_Master\_Receive\_IT()*
- *HAL\_FMPI2C\_Slave\_Transmit\_IT()*
- *HAL\_FMPI2C\_Slave\_Receive\_IT()*
- *HAL\_FMPI2C\_Master\_Transmit\_DMA()*
- *HAL\_FMPI2C\_Master\_Receive\_DMA()*
- *HAL\_FMPI2C\_Slave\_Transmit\_DMA()*
- *HAL\_FMPI2C\_Slave\_Receive\_DMA()*
- *HAL\_FMPI2C\_Mem\_Write()*
- *HAL\_FMPI2C\_Mem\_Read()*
- *HAL\_FMPI2C\_Mem\_Write\_IT()*
- *HAL\_FMPI2C\_Mem\_Read\_IT()*
- *HAL\_FMPI2C\_Mem\_Write\_DMA()*
- *HAL\_FMPI2C\_Mem\_Read\_DMA()*
- *HAL\_FMPI2C\_IsDeviceReady()*
- *HAL\_FMPI2C\_Master\_Seq\_Transmit\_IT()*
- *HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA()*
- *HAL\_FMPI2C\_Master\_Seq\_Receive\_IT()*
- *HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA()*
- *HAL\_FMPI2C\_Slave\_Seq\_Transmit\_IT()*
- *HAL\_FMPI2C\_Slave\_Seq\_Transmit\_DMA()*
- *HAL\_FMPI2C\_Slave\_Seq\_Receive\_IT()*
- *HAL\_FMPI2C\_Slave\_Seq\_Receive\_DMA()*
- *HAL\_FMPI2C\_EnableListen\_IT()*
- *HAL\_FMPI2C\_DisableListen\_IT()*
- *HAL\_FMPI2C\_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\_FMPI2C\_GetState()*
- *HAL\_FMPI2C\_GetMode()*
- *HAL\_FMPI2C\_GetError()*

## 29.2.5 Detailed description of functions

### HAL\_FMPI2C\_Init

#### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Init (FMPI2C\_HandleTypeDef \* hfmpi2c)**

#### Function description

Initializes the FMPI2C according to the specified parameters in the FMPI2C\_InitTypeDef and initialize the associated handle.

#### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **HAL**: status

### HAL\_FMPI2C\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_DeInit (FMPI2C\_HandleTypeDef \* hfmpi2c)**

#### Function description

DeInitialize the FMPI2C peripheral.

#### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **HAL**: status

### HAL\_FMPI2C\_Msplnit

#### Function name

**void HAL\_FMPI2C\_Msplnit (FMPI2C\_HandleTypeDef \* hfmpi2c)**

#### Function description

Initialize the FMPI2C MSP.

#### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None**:

### HAL\_FMPI2C\_MspDeInit

#### Function name

**void HAL\_FMPI2C\_MspDeInit (FMPI2C\_HandleTypeDef \* hfmpi2c)**

#### Function description

DeInitialize the FMPI2C MSP.

#### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **None:**

**HAL\_FMPI2C\_Master\_Transmit**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Transmit (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Receive**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Receive (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Slave\_Transmit**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Transmit (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

### Function description

Transmits in slave mode an amount of data in blocking mode.

### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

#### HAL\_FMPI2C\_Slave\_Receive

### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Receive (FMPI2C\_HandleTypeDef \* hfmpi2c, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)

### Function description

Receive in slave mode an amount of data in blocking mode.

### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

#### HAL\_FMPI2C\_Mem\_Write

### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Mem\_Write (FMPI2C\_HandleTypeDef \* hfmpi2c, 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

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Mem\_Read

### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Mem\_Read (FMPI2C\_HandleTypeDef \* hfmpi2c, 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

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_IsDeviceReady**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_IsDeviceReady (FMPI2C\_HandleTypeDef \* hfmpi2c, uint16\_t DevAddress, uint32\_t Trials, uint32\_t Timeout)**

### Function description

Checks if target device is ready for communication.

### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Transmit\_IT**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Transmit\_IT (FMPI2C\_HandleTypeDef \* hfmpi2c, 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

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Receive\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Receive\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Slave\_Transmit\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Transmit\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t \* pData, uint16\_t Size)**

### Function description

Transmit in slave mode an amount of data in non-blocking mode with Interrupt.

### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

### Return values

- **HAL:** status

### HAL\_FMPI2C\_Slave\_Receive\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Receive\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t \* pData, uint16\_t Size)**

### Function description

Receive in slave mode an amount of data in non-blocking mode with Interrupt.

### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

### Return values

- **HAL:** status

**HAL\_FMPI2C\_Mem\_Write\_IT**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Mem\_Write\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Mem\_Read\_IT**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Mem\_Read\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Seq\_Transmit\_IT**

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Seq\_Transmit\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, uint16\_t DevAddress, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

### Function description

Sequential transmit in master FMPI2C mode an amount of data in non-blocking mode with Interrupt.



### Parameters

- **hfmipi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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 FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

#### HAL\_FMPI2C\_Master\_Seq\_Receive\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Seq\_Receive\_IT (FMPI2C\_HandleTypeDef \* hfmipi2c, uint16\_t DevAddress, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

### Function description

Sequential receive in master FMPI2C mode an amount of data in non-blocking mode with Interrupt.

### Parameters

- **hfmipi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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 FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

#### HAL\_FMPI2C\_Slave\_Seq\_Transmit\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Seq\_Transmit\_IT (FMPI2C\_HandleTypeDef \* hfmipi2c, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

### Function description

Sequential transmit in slave/device FMPI2C mode an amount of data in non-blocking mode with Interrupt.

### Parameters

- **hfmipi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

#### HAL\_FMPI2C\_Slave\_Seq\_Receive\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Seq\_Receive\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

### Function description

Sequential receive in slave/device FMPI2C mode an amount of data in non-blocking mode with Interrupt.

### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **XferOptions**: Options of Transfer, value of FMPI2C Sequential Transfer Options

### Return values

- **HAL**: status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

#### HAL\_FMPI2C\_EnableListen\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_EnableListen\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c)**

### Function description

Enable the Address listen mode with Interrupt.

### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **HAL**: status

#### HAL\_FMPI2C\_DisableListen\_IT

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_DisableListen\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c)**

### Function description

Disable the Address listen mode with Interrupt.

### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C

### Return values

- **HAL**: status

## HAL\_FMPI2C\_Master\_Abort\_IT

### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Abort\_IT (FMPI2C\_HandleTypeDef \* hfmapi2c, uint16\_t DevAddress)

### Function description

Abort a master FMPI2C IT or DMA process communication with Interrupt.

### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Transmit\_DMA

### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Transmit\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Receive\_DMA

### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Receive\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Slave\_Transmit\_DMA

#### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Transmit\_DMA (FMPI2C\_HandleTypeDef \* hfmpi2c, uint8\_t \* pData, uint16\_t Size)

#### Function description

Transmit in slave mode an amount of data in non-blocking mode with DMA.

#### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

#### Return values

- **HAL:** status

### HAL\_FMPI2C\_Slave\_Receive\_DMA

#### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Receive\_DMA (FMPI2C\_HandleTypeDef \* hfmpi2c, uint8\_t \* pData, uint16\_t Size)

#### Function description

Receive in slave mode an amount of data in non-blocking mode with DMA.

#### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

#### Return values

- **HAL:** status

### HAL\_FMPI2C\_Mem\_Write\_DMA

#### Function name

HAL\_StatusTypeDef HAL\_FMPI2C\_Mem\_Write\_DMA (FMPI2C\_HandleTypeDef \* hfmpi2c, 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

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Mem\_Read\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Mem\_Read\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, 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

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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\_FMPI2C\_Master\_Seq\_Transmit\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Seq\_Transmit\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, uint16\_t DevAddress, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

### Function description

Sequential transmit in master FMPI2C mode an amount of data in non-blocking mode with DMA.

### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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 FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

### HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Master\_Seq\_Receive\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, uint16\_t DevAddress, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

### Function description

Sequential receive in master FMPI2C mode an amount of data in non-blocking mode with DMA.

### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **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 FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

### HAL\_FMPI2C\_Slave\_Seq\_Transmit\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Seq\_Transmit\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

#### Function description

Sequential transmit in slave/device FMPI2C mode an amount of data in non-blocking mode with DMA.

### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

### Notes

- This interface allow to manage repeated start condition when a direction change during transfer

### HAL\_FMPI2C\_Slave\_Seq\_Receive\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_FMPI2C\_Slave\_Seq\_Receive\_DMA (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t \* pData, uint16\_t Size, uint32\_t XferOptions)**

#### Function description

Sequential receive in slave/device FMPI2C mode an amount of data in non-blocking mode with DMA.

### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of FMPI2C Sequential Transfer Options

### Return values

- **HAL:** status

## Notes

- This interface allow to manage repeated start condition when a direction change during transfer

### HAL\_FMPI2C\_EV\_IRQHandler

#### Function name

**void HAL\_FMPI2C\_EV\_IRQHandler (FMPI2C\_HandleTypeDef \* hfmapi2c)**

#### Function description

This function handles FMPI2C event interrupt request.

#### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None**:

### HAL\_FMPI2C\_ER\_IRQHandler

#### Function name

**void HAL\_FMPI2C\_ER\_IRQHandler (FMPI2C\_HandleTypeDef \* hfmapi2c)**

#### Function description

This function handles FMPI2C error interrupt request.

#### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None**:

### HAL\_FMPI2C\_MasterTxCpltCallback

#### Function name

**void HAL\_FMPI2C\_MasterTxCpltCallback (FMPI2C\_HandleTypeDef \* hfmapi2c)**

#### Function description

Master Tx Transfer completed callback.

#### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None**:

### HAL\_FMPI2C\_MasterRxCpltCallback

#### Function name

**void HAL\_FMPI2C\_MasterRxCpltCallback (FMPI2C\_HandleTypeDef \* hfmapi2c)**

#### Function description

Master Rx Transfer completed callback.

#### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None:**

**HAL\_FMPI2C\_SlaveTxCpltCallback**

#### Function name

**void HAL\_FMPI2C\_SlaveTxCpltCallback (FMPI2C\_HandleTypeDef \* hfmapi2c)**

#### Function description

Slave Tx Transfer completed callback.

#### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None:**

**HAL\_FMPI2C\_SlaveRxCpltCallback**

#### Function name

**void HAL\_FMPI2C\_SlaveRxCpltCallback (FMPI2C\_HandleTypeDef \* hfmapi2c)**

#### Function description

Slave Rx Transfer completed callback.

#### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

#### Return values

- **None:**

**HAL\_FMPI2C\_AddrCallback**

#### Function name

**void HAL\_FMPI2C\_AddrCallback (FMPI2C\_HandleTypeDef \* hfmapi2c, uint8\_t TransferDirection, uint16\_t AddrMatchCode)**

#### Function description

Slave Address Match callback.

#### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of FMPI2C Transfer Direction Master Point of View
- **AddrMatchCode:** Address Match Code

#### Return values

- **None:**

**HAL\_FMPI2C\_ListenCpltCallback**

#### Function name

**void HAL\_FMPI2C\_ListenCpltCallback (FMPI2C\_HandleTypeDef \* hfmapi2c)**



### Function description

Listen Complete callback.

### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **None:**

**HAL\_FMPI2C\_MemTxCpltCallback**

### Function name

**void HAL\_FMPI2C\_MemTxCpltCallback (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

Memory Tx Transfer completed callback.

### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **None:**

**HAL\_FMPI2C\_MemRxCpltCallback**

### Function name

**void HAL\_FMPI2C\_MemRxCpltCallback (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

Memory Rx Transfer completed callback.

### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **None:**

**HAL\_FMPI2C\_ErrorCallback**

### Function name

**void HAL\_FMPI2C\_ErrorCallback (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

FMPI2C error callback.

### Parameters

- **hfmpi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **None:**

**HAL\_FMPI2C\_AbortCpltCallback**

### Function name

**void HAL\_FMPI2C\_AbortCpltCallback (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

FMPI2C abort callback.

### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **None**:

**HAL\_FMPI2C\_GetState**

### Function name

**HAL\_FMPI2C\_StateTypeDef HAL\_FMPI2C\_GetState (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

Return the FMPI2C handle state.

### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **HAL**: state

**HAL\_FMPI2C\_GetMode**

### Function name

**HAL\_FMPI2C\_ModeTypeDef HAL\_FMPI2C\_GetMode (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

Returns the FMPI2C Master, Slave, Memory or no mode.

### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for FMPI2C module

### Return values

- **HAL**: mode

**HAL\_FMPI2C\_GetError**

### Function name

**uint32\_t HAL\_FMPI2C\_GetError (FMPI2C\_HandleTypeDef \* hfmpi2c)**

### Function description

Return the FMPI2C error code.

### Parameters

- **hfmpi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2C.

### Return values

- **FMPI2C**: Error Code

## 29.3 FMPI2C Firmware driver defines

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

### 29.3.1 FMPI2C

FMPI2C  
*FMPI2C Addressing Mode*

FMPI2C\_ADDRESSINGMODE\_7BIT

FMPI2C\_ADDRESSINGMODE\_10BIT

*FMPI2C Dual Addressing Mode*

FMPI2C\_DUALADDRESS\_DISABLE

FMPI2C\_DUALADDRESS\_ENABLE

*FMPI2C Error Code definition*

HAL\_FMPI2C\_ERROR\_NONE

No error

HAL\_FMPI2C\_ERROR\_BERR

BERR error

HAL\_FMPI2C\_ERROR\_ARLO

ARLO error

HAL\_FMPI2C\_ERROR\_AF

ACKF error

HAL\_FMPI2C\_ERROR\_OVR

OVR error

HAL\_FMPI2C\_ERROR\_DMA

DMA transfer error

HAL\_FMPI2C\_ERROR\_TIMEOUT

Timeout error

HAL\_FMPI2C\_ERROR\_SIZE

Size Management error

HAL\_FMPI2C\_ERROR\_DMA\_PARAM

DMA Parameter Error

HAL\_FMPI2C\_ERROR\_INVALID\_PARAM

Invalid Parameters error

*FMPI2C Exported Macros*

**\_\_HAL\_FMPI2C\_RESET\_HANDLE\_STATE**

**Description:**

- Reset FMPI2C handle state.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.

**Return value:**

- None

### **\_\_HAL\_FMPI2C\_ENABLE\_IT**

**Description:**

- Enable the specified FMPI2C interrupt.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable. This parameter can be one of the following values:
  - `FMPI2C_IT_ERRI` Errors interrupt enable
  - `FMPI2C_IT_TCI` Transfer complete interrupt enable
  - `FMPI2C_IT_STOPI` STOP detection interrupt enable
  - `FMPI2C_IT_NACKI` NACK received interrupt enable
  - `FMPI2C_IT_ADDRI` Address match interrupt enable
  - `FMPI2C_IT_RXI` RX interrupt enable
  - `FMPI2C_IT_TXI` TX interrupt enable

**Return value:**

- None

### **\_\_HAL\_FMPI2C\_DISABLE\_IT**

**Description:**

- Disable the specified FMPI2C interrupt.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to disable. This parameter can be one of the following values:
  - `FMPI2C_IT_ERRI` Errors interrupt enable
  - `FMPI2C_IT_TCI` Transfer complete interrupt enable
  - `FMPI2C_IT_STOPI` STOP detection interrupt enable
  - `FMPI2C_IT_NACKI` NACK received interrupt enable
  - `FMPI2C_IT_ADDRI` Address match interrupt enable
  - `FMPI2C_IT_RXI` RX interrupt enable
  - `FMPI2C_IT_TXI` TX interrupt enable

**Return value:**

- None

### **\_\_HAL\_FMPI2C\_GET\_IT\_SOURCE**

**Description:**

- Check whether the specified FMPI2C interrupt source is enabled or not.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.
- `__INTERRUPT__`: specifies the FMPI2C interrupt source to check. This parameter can be one of the following values:
  - `FMPI2C_IT_ERRI` Errors interrupt enable
  - `FMPI2C_IT_TCI` Transfer complete interrupt enable
  - `FMPI2C_IT_STOPI` STOP detection interrupt enable
  - `FMPI2C_IT_NACKI` NACK received interrupt enable
  - `FMPI2C_IT_ADDRI` Address match interrupt enable
  - `FMPI2C_IT_RXI` RX interrupt enable
  - `FMPI2C_IT_TXI` TX interrupt enable

**Return value:**

- The: new state of `__INTERRUPT__` (SET or RESET).

## FMPI2C\_FLAG\_MASK

**Description:**

- Check whether the specified FMPI2C flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `FMPI2C_FLAG_TXE` Transmit data register empty
  - `FMPI2C_FLAG_TXIS` Transmit interrupt status
  - `FMPI2C_FLAG_RXNE` Receive data register not empty
  - `FMPI2C_FLAG_ADDR` Address matched (slave mode)
  - `FMPI2C_FLAG_AF` Acknowledge failure received flag
  - `FMPI2C_FLAG_STOPF` STOP detection flag
  - `FMPI2C_FLAG_TC` Transfer complete (master mode)
  - `FMPI2C_FLAG_TCR` Transfer complete reload
  - `FMPI2C_FLAG_BERR` Bus error
  - `FMPI2C_FLAG_ARLO` Arbitration lost
  - `FMPI2C_FLAG_OVR` Overrun/Underrun
  - `FMPI2C_FLAG_PECERR` PEC error in reception
  - `FMPI2C_FLAG_TIMEOUT` Timeout or Tlow detection flag
  - `FMPI2C_FLAG_ALERT` SMBus alert
  - `FMPI2C_FLAG_BUSY` Bus busy
  - `FMPI2C_FLAG_DIR` Transfer direction (slave mode)

**Return value:**

- The: new state of `__FLAG__` (SET or RESET).

## `__HAL_FMPI2C_GET_FLAG`

## `__HAL_FMPI2C_CLEAR_FLAG`

**Description:**

- Clear the FMPI2C pending flags which are cleared by writing 1 in a specific bit.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `FMPI2C_FLAG_TXE` Transmit data register empty
  - `FMPI2C_FLAG_ADDR` Address matched (slave mode)
  - `FMPI2C_FLAG_AF` Acknowledge failure received flag
  - `FMPI2C_FLAG_STOPF` STOP detection flag
  - `FMPI2C_FLAG_BERR` Bus error
  - `FMPI2C_FLAG_ARLO` Arbitration lost
  - `FMPI2C_FLAG_OVR` Overrun/Underrun
  - `FMPI2C_FLAG_PECERR` PEC error in reception
  - `FMPI2C_FLAG_TIMEOUT` Timeout or Tlow detection flag
  - `FMPI2C_FLAG_ALERT` SMBus alert

**Return value:**

- None

**\_\_HAL\_FMPI2C\_ENABLE****Description:**

- Enable the specified FMPI2C peripheral.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.

**Return value:**

- None

**\_\_HAL\_FMPI2C\_DISABLE****Description:**

- Disable the specified FMPI2C peripheral.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.

**Return value:**

- None

**\_\_HAL\_FMPI2C\_GENERATE\_NACK****Description:**

- Generate a Non-Acknowledge FMPI2C peripheral in Slave mode.

**Parameters:**

- `__HANDLE__`: specifies the FMPI2C Handle.

**Return value:**

- None

***FMPI2C Flag definition***

FMPI2C\_FLAG\_TXE

FMPI2C\_FLAG\_TXIS

FMPI2C\_FLAG\_RXNE

FMPI2C\_FLAG\_ADDR

FMPI2C\_FLAG\_AF

FMPI2C\_FLAG\_STOPF

FMPI2C\_FLAG\_TC

FMPI2C\_FLAG\_TCR

FMPI2C\_FLAG\_BERR

FMPI2C\_FLAG\_ARLO

FMPI2C\_FLAG\_OVR

FMPI2C\_FLAG\_PECERR

FMPI2C\_FLAG\_TIMEOUT

FMPI2C\_FLAG\_ALERT

FMPI2C\_FLAG\_BUSY

FMPI2C\_FLAG\_DIR

*FMPI2C General Call Addressing Mode*

FMPI2C\_GENERALCALL\_DISABLE

FMPI2C\_GENERALCALL\_ENABLE

*FMPI2C Interrupt configuration definition*

FMPI2C\_IT\_ERRI

FMPI2C\_IT\_TCI

FMPI2C\_IT\_STOPI

FMPI2C\_IT\_NACKI

FMPI2C\_IT\_ADDRI

FMPI2C\_IT\_RXI

FMPI2C\_IT\_TXI

*FMPI2C Memory Address Size*

FMPI2C\_MEMADD\_SIZE\_8BIT

FMPI2C\_MEMADD\_SIZE\_16BIT

*FMPI2C No-Stretch Mode*

FMPI2C\_NOSTRETCH\_DISABLE

FMPI2C\_NOSTRETCH\_ENABLE

*FMPI2C Own Address2 Masks*

FMPI2C\_OA2\_NOMASK

FMPI2C\_OA2\_MASK01

FMPI2C\_OA2\_MASK02

FMPI2C\_OA2\_MASK03

FMPI2C\_OA2\_MASK04

FMPI2C\_OA2\_MASK05

FMPI2C\_OA2\_MASK06

FMPI2C\_OA2\_MASK07

*FMPI2C Reload End Mode*

FMPI2C\_RELOAD\_MODE

FMPI2C\_AUTOEND\_MODE

FMPI2C\_SOFTEND\_MODE

*FMPI2C Start or Stop Mode*

FMPI2C\_NO\_STARTSTOP

FMPI2C\_GENERATE\_STOP

FMPI2C\_GENERATE\_START\_READ

FMPI2C\_GENERATE\_START\_WRITE

*FMPI2C Transfer Direction Master Point of View*

FMPI2C\_DIRECTION\_TRANSMIT

FMPI2C\_DIRECTION\_RECEIVE

*FMPI2C Sequential Transfer Options*

FMPI2C\_FIRST\_FRAME

FMPI2C\_FIRST\_AND\_NEXT\_FRAME

FMPI2C\_NEXT\_FRAME

FMPI2C\_FIRST\_AND\_LAST\_FRAME

FMPI2C\_LAST\_FRAME

FMPI2C\_LAST\_FRAME\_NO\_STOP

FMPI2C\_OTHER\_FRAME

FMPI2C\_OTHER\_AND\_LAST\_FRAME



## 30 HAL FMPI2C Extension Driver

### 30.1 FMPI2CEx Firmware driver API description

The following section lists the various functions of the FMPI2CEx library.

#### 30.1.1 FMPI2C peripheral Extended features

Comparing to other previous devices, the FMPI2C interface for STM32F4xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable Fast Mode Plus

#### 30.1.2 How to use this driver

This driver provides functions to:

1. Configure FMPI2C Analog noise filter using the function `HAL_FMPI2CEx_ConfigAnalogFilter()`
2. Configure FMPI2C Digital noise filter using the function `HAL_FMPI2CEx_ConfigDigitalFilter()`
3. Configure the enable or disable of fast mode plus driving capability using the functions :
  - `HAL_FMPI2CEx_EnableFastModePlus()`
  - `HAL_FMPI2CEx_DisableFastModePlus()`

#### 30.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Fast Mode Plus

This section contains the following APIs:

- [\*HAL\\_FMPI2CEx\\_ConfigAnalogFilter\(\)\*](#)
- [\*HAL\\_FMPI2CEx\\_ConfigDigitalFilter\(\)\*](#)
- [\*HAL\\_FMPI2CEx\\_EnableFastModePlus\(\)\*](#)
- [\*HAL\\_FMPI2CEx\\_DisableFastModePlus\(\)\*](#)

#### 30.1.4 Detailed description of functions

##### HAL\_FMPI2CEx\_ConfigAnalogFilter

###### Function name

`HAL_StatusTypeDef HAL_FMPI2CEx_ConfigAnalogFilter (FMPI2C_HandleTypeDef * hfmapi2c, uint32_t AnalogFilter)`

###### Function description

Configure FMPI2C Analog noise filter.

###### Parameters

- **hfmapi2c**: Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2Cx peripheral.
- **AnalogFilter**: New state of the Analog filter.

###### Return values

- **HAL**: status

### HAL\_FMPI2CEx\_ConfigDigitalFilter

#### Function name

**HAL\_StatusTypeDef HAL\_FMPI2CEx\_ConfigDigitalFilter (FMPI2C\_HandleTypeDef \* hfmapi2c, uint32\_t DigitalFilter)**

#### Function description

Configure FMPI2C Digital noise filter.

#### Parameters

- **hfmapi2c:** Pointer to a FMPI2C\_HandleTypeDef structure that contains the configuration information for the specified FMPI2Cx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between Min\_Data=0x00 and Max\_Data=0x0F.

#### Return values

- **HAL:** status

### HAL\_FMPI2CEx\_EnableFastModePlus

#### Function name

**void HAL\_FMPI2CEx\_EnableFastModePlus (uint32\_t ConfigFastModePlus)**

#### Function description

Enable the FMPI2C fast mode plus driving capability.

#### Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the FMPI2C Extended Fast Mode Plus values

#### Return values

- **None:**

#### Notes

- For FMPI2C1, fast mode plus driving capability can be enabled on all selected FMPI2C1 pins using FMPI2C\_FASTMODEPLUS\_FMPI2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining FMPI2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using FMPI2C\_FASTMODEPLUS\_FMPI2C1 parameter.

### HAL\_FMPI2CEx\_DisableFastModePlus

#### Function name

**void HAL\_FMPI2CEx\_DisableFastModePlus (uint32\_t ConfigFastModePlus)**

#### Function description

Disable the FMPI2C fast mode plus driving capability.

#### Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the FMPI2C Extended Fast Mode Plus values

#### Return values

- **None:**

**Notes**

- For FMPI2C1, fast mode plus driving capability can be disabled on all selected FMPI2C1 pins using FMPI2C\_FASTMODEPLUS\_FMPI2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining FMPI2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using FMPI2C\_FASTMODEPLUS\_FMPI2C1 parameter.

## 30.2 FMPI2CEx Firmware driver defines

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

### 30.2.1 FMPI2CEx

FMPI2CEx

*FMPI2C Extended Analog Filter*

**FMPI2C\_ANALOGFILTER\_ENABLE**

**FMPI2C\_ANALOGFILTER\_DISABLE**

*FMPI2C Extended Fast Mode Plus*

**FMPI2C\_FASTMODEPLUS\_SCL**

Enable Fast Mode Plus on FMPI2C1 SCL pins

**FMPI2C\_FASTMODEPLUS\_SDA**

Enable Fast Mode Plus on FMPI2C1 SDA pins

## 31 HAL GPIO Generic Driver

### 31.1 GPIO Firmware driver registers structures

#### 31.1.1 GPIO\_InitTypeDef

*GPIO\_InitTypeDef* is defined in the `stm32f4xx_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\\_define](#)
- *uint32\_t GPIO\_InitTypeDef::Mode*  
Specifies the operating mode for the selected pins. This parameter can be a value of [GPIO\\_mode\\_define](#)
- *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\\_define](#)
- *uint32\_t GPIO\_InitTypeDef::Speed*  
Specifies the speed for the selected pins. This parameter can be a value of [GPIO\\_speed\\_define](#)
- *uint32\_t GPIO\_InitTypeDef::Alternate*  
Peripheral to be connected to the selected pins. This parameter can be a value of [GPIO\\_Alternate\\_function\\_selection](#)

### 31.2 GPIO Firmware driver API description

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

#### 31.2.1 GPIO Peripheral features

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the General Purpose IO (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.

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 23 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.

#### 31.2.2 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 PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

### 31.2.3 Initialization and de-initialization functions

This section provides functions allowing to initialize and de-initialize the GPIOs to be ready for use.

This section contains the following APIs:

- [\*HAL\\_GPIO\\_Init\(\)\*](#)
- [\*HAL\\_GPIO\\_DeInit\(\)\*](#)

### 31.2.4 IO operation functions

This section contains the following APIs:

- [\*HAL\\_GPIO\\_ReadPin\(\)\*](#)
- [\*HAL\\_GPIO\\_WritePin\(\)\*](#)
- [\*HAL\\_GPIO\\_TogglePin\(\)\*](#)
- [\*HAL\\_GPIO\\_LockPin\(\)\*](#)
- [\*HAL\\_GPIO\\_EXTI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_GPIO\\_EXTI\\_Callback\(\)\*](#)

### 31.2.5 Detailed description of functions

#### HAL\_GPIO\_Init

##### Function name

**void HAL\_GPIO\_Init (GPIO\_TypeDef \* GPIOx, GPIO\_InitTypeDef \* GPIO\_Init)**

##### Function description

Initializes the GPIOx peripheral according to the specified parameters in the `GPIO_Init`.

##### Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
- **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-initializes the GPIOx peripheral registers to their default reset values.

**Parameters**

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
- **GPIO\_Pin:** specifies the port bit to be written. This parameter can be one 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**

Reads the specified input port pin.

**Parameters**

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
- **GPIO\_Pin:** specifies the port bit to read. This parameter can be 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**

Sets or clears the selected data port bit.

**Parameters**

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
- **GPIO\_Pin:** specifies the port bit to be written. This parameter can be one 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 register 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

Toggles the specified GPIO pins.

#### Parameters

- **GPIOx:** Where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
- **GPIO\_Pin:** Specifies the pins to be toggled.

#### Return values

- **None:**

### HAL\_GPIO\_LockPin

#### Function name

**HAL\_StatusTypeDef HAL\_GPIO\_LockPin (GPIO\_TypeDef \* GPIOx, uint16\_t GPIO\_Pin)**

#### Function description

Locks GPIO Pins configuration registers.

#### Parameters

- **GPIOx:** where x can be (A..F) to select the GPIO peripheral for STM32F4 family
- **GPIO\_Pin:** specifies the port bit 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\_AFR1 and GPIOx\_AFR2.
- 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

This function handles EXTI interrupt request.

#### Parameters

- **GPIO\_Pin:** Specifies the pins connected 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 callbacks.

### Parameters

- **GPIO\_Pin:** Specifies the pins connected EXTI line

### Return values

- **None:**

## 31.3 GPIO Firmware driver defines

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

### 31.3.1 GPIO

GPIO

***GPIO Alternate Function Selection***

GPIO\_AF0\_RTC\_50Hz

GPIO\_AF0\_MCO

GPIO\_AF0\_TAMPER

GPIO\_AF0\_SWJ

GPIO\_AF0\_TRACE

GPIO\_AF1\_TIM1

GPIO\_AF1\_TIM2

GPIO\_AF2\_TIM3

GPIO\_AF2\_TIM4

GPIO\_AF2\_TIM5

GPIO\_AF3\_TIM8

GPIO\_AF3\_TIM9

GPIO\_AF3\_TIM10

GPIO\_AF3\_TIM11

GPIO\_AF4\_I2C1

GPIO\_AF4\_I2C2

GPIO\_AF4\_I2C3



GPIO\_AF5\_SPI1

GPIO\_AF5\_SPI2

GPIO\_AF5\_SPI3

GPIO\_AF5\_SPI4

GPIO\_AF5\_SPI5

GPIO\_AF5\_SPI6

GPIO\_AF5\_I2S3ext

GPIO\_AF6\_SPI3

GPIO\_AF6\_I2S2ext

GPIO\_AF6\_SAI1

GPIO\_AF7\_USART1

GPIO\_AF7\_USART2

GPIO\_AF7\_USART3

GPIO\_AF7\_I2S3ext

GPIO\_AF8\_UART4

GPIO\_AF8\_UART5

GPIO\_AF8\_USART6

GPIO\_AF8\_UART7

GPIO\_AF8\_UART8

GPIO\_AF9\_CAN1

GPIO\_AF9\_CAN2

GPIO\_AF9\_TIM12

GPIO\_AF9\_TIM13

GPIO\_AF9\_TIM14

GPIO\_AF9\_LTDC

GPIO\_AF9\_QSPI

GPIO\_AF10\_OTG\_FS

GPIO\_AF10\_OTG\_HS

GPIO\_AF10\_QSPI

GPIO\_AF11\_ETH

GPIO\_AF12\_FMC

GPIO\_AF12\_OTG\_HS\_FS

GPIO\_AF12\_SDIO

GPIO\_AF13\_DCMI

GPIO\_AF13\_DSI

GPIO\_AF14\_LTDC

GPIO\_AF15\_EVENTOUT

### *GPIO Exported Macros*

**\_\_HAL\_GPIO\_EXTI\_GET\_FLAG**

**Description:**

- Checks 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:**

- Clears 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:**

- Checks 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:**

- Clears 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:**

- Generates 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 define**

### **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 define**

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 define***

GPIO\_NOPULL

No Pull-up or Pull-down activation

GPIO\_PULLUP

Pull-up activation

GPIO\_PULLDOWN

Pull-down activation

***GPIO speed define***

GPIO\_SPEED\_FREQ\_LOW

IO works at 2 MHz, please refer to the product datasheet

GPIO\_SPEED\_FREQ\_MEDIUM

range 12,5 MHz to 50 MHz, please refer to the product datasheet

GPIO\_SPEED\_FREQ\_HIGH

range 25 MHz to 100 MHz, please refer to the product datasheet

**GPIO\_SPEED\_FREQ\_VERY\_HIGH**

range 50 MHz to 200 MHz, please refer to the product datasheet

---

## 32 HAL GPIO Extension Driver

---

### 32.1 GPIOEx Firmware driver defines

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

#### 32.1.1 GPIOEx

GPIOEx

*GPIO Get Port Index*

#### GPIO\_GET\_INDEX

*GPIO Check Alternate Function*

#### IS\_GPIO\_AF

## 33 HAL HASH Generic Driver

### 33.1 HASH Firmware driver registers structures

#### 33.1.1 HASH\_InitTypeDef

*HASH\_InitTypeDef* is defined in the stm32f4xx\_hal\_hash.h

##### Data Fields

- *uint32\_t* *DataType*
- *uint32\_t* *KeySize*
- *uint8\_t* \* *pKey*

##### Field Documentation

- *uint32\_t* *HASH\_InitTypeDef::DataType*  
32-bit data, 16-bit data, 8-bit data or 1-bit data. This parameter can be a value of *HASH\_Data\_Type*.
- *uint32\_t* *HASH\_InitTypeDef::KeySize*  
The key size is used only in HMAC operation.
- *uint8\_t*\* *HASH\_InitTypeDef::pKey*  
The key is used only in HMAC operation.

#### 33.1.2 HASH\_HandleTypeDef

*HASH\_HandleTypeDef* is defined in the stm32f4xx\_hal\_hash.h

##### Data Fields

- *HASH\_InitTypeDef* *Init*
- *uint8\_t* \* *pHashInBuffPtr*
- *uint8\_t* \* *pHashOutBuffPtr*
- *uint8\_t* \* *pHashKeyBuffPtr*
- *uint8\_t* \* *pHashMsgBuffPtr*
- *uint32\_t* *HashBuffSize*
- *\_\_IO uint32\_t* *HashInCount*
- *\_\_IO uint32\_t* *HashITCounter*
- *\_\_IO uint32\_t* *HashKeyCount*
- *HAL\_StatusTypeDef* *Status*
- *HAL\_HASH\_PhaseTypeDef* *Phase*
- *DMA\_HandleTypeDef* \* *hdmain*
- *HAL\_LockTypeDef* *Lock*
- *\_\_IO HAL\_HASH\_StateTypeDef* *State*
- *HAL\_HASH\_SuspendTypeDef* *SuspendRequest*
- *FlagStatus* *DigestCalculationDisable*
- *\_\_IO uint32\_t* *NbWordsAlreadyPushed*
- *\_\_IO uint32\_t* *ErrorCode*
- *\_\_IO uint32\_t* *Accumulation*

##### Field Documentation

- *HASH\_InitTypeDef* *HASH\_HandleTypeDef::Init*  
HASH required parameters
- *uint8\_t*\* *HASH\_HandleTypeDef::pHashInBuffPtr*  
Pointer to input buffer
- *uint8\_t*\* *HASH\_HandleTypeDef::pHashOutBuffPtr*  
Pointer to output buffer (digest)

- ***uint8\_t\* HASH\_HandleTypeDef::pHashKeyBuffPtr***  
Pointer to key buffer (HMAC only)
- ***uint8\_t\* HASH\_HandleTypeDef::pHashMsgBuffPtr***  
Pointer to message buffer (HMAC only)
- ***uint32\_t HASH\_HandleTypeDef::HashBuffSize***  
Size of buffer to be processed
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::HashInCount***  
Counter of inputted data
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::HashITCounter***  
Counter of issued interrupts
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::HashKeyCount***  
Counter for Key inputted data (HMAC only)
- ***HAL\_StatusTypeDef HASH\_HandleTypeDef::Status***  
HASH peripheral status
- ***HAL\_HASH\_PhaseTypeDef HASH\_HandleTypeDef::Phase***  
HASH peripheral phase
- ***DMA\_HandleTypeDef\* HASH\_HandleTypeDef::hdmain***  
HASH In DMA Handle parameters
- ***HAL\_LockTypeDef HASH\_HandleTypeDef::Lock***  
Locking object
- ***\_\_IO HAL\_HASH\_StateTypeDef HASH\_HandleTypeDef::State***  
HASH peripheral state
- ***HAL\_HASH\_SuspendTypeDef HASH\_HandleTypeDef::SuspendRequest***  
HASH peripheral suspension request flag
- ***FlagStatus HASH\_HandleTypeDef::DigestCalculationDisable***  
Digest calculation phase skip (MDMAT bit control) for multi-buffers DMA-based HMAC computation
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::NbWordsAlreadyPushed***  
Numbers of words already pushed in FIFO before inputting new block
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::ErrorCode***  
HASH Error code
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::Accumulation***  
HASH multi buffers accumulation flag

## 33.2 HASH Firmware driver API description

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

### 33.2.1 How to use this driver

The HASH HAL driver can be used as follows:



1. Initialize the HASH low level resources by implementing the HAL\_HASH\_MspInit():
  - a. Enable the HASH interface clock using `__HASH_CLK_ENABLE()`
  - b. When resorting to interrupt-based APIs (e.g. `HAL_HASH_xxx_Start_IT()`)
    - Configure the HASH interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the HASH IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In HASH IRQ handler, call `HAL_HASH_IRQHandler()` API
  - c. When resorting to DMA-based APIs (e.g. `HAL_HASH_xxx_Start_DMA()`)
    - Enable the DMAx interface clock using `__DMAx_CLK_ENABLE()`
    - Configure and enable one DMA stream to manage data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU.
    - Associate the initialized DMA handle to the HASH DMA handle using `__HAL_LINKDMA()`
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Stream: use `HAL_NVIC_SetPriority()` and `HAL_NVIC_EnableIRQ()`
2. Initialize the HASH HAL using `HAL_HASH_Init()`. This function:
  - a. resorts to `HAL_HASH_MspInit()` for low-level initialization,
  - b. configures the data type: 1-bit, 8-bit, 16-bit or 32-bit.
3. Three processing schemes are available:
  - a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. `HAL_HASH_xxx_Start()` for HASH or `HAL_HMAC_xxx_Start()` for HMAC
  - b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. `HAL_HASH_xxx_Start_IT()` for HASH or `HAL_HMAC_xxx_Start_IT()` for HMAC
  - c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. `HAL_HASH_xxx_Start_DMA()` for HASH or `HAL_HMAC_xxx_Start_DMA()` for HMAC. Note that in DMA mode, a call to `HAL_HASH_xxx_Finish()` is then required to retrieve the digest.
4. When the processing function is called after `HAL_HASH_Init()`, the HASH peripheral is initialized and processes the buffer fed in input. When the input data have all been fed to the Peripheral, the digest computation can start.
5. Multi-buffer processing is possible in polling, interrupt and DMA modes.
  - a. In polling mode, only multi-buffer HASH processing is possible. API `HAL_HASH_xxx_Accumulate()` must be called for each input buffer, except for the last one. User must resort to `HAL_HASH_xxx_Accumulate_End()` to enter the last one and retrieve as well the computed digest.
  - b. In interrupt mode, API `HAL_HASH_xxx_Accumulate_IT()` must be called for each input buffer, except for the last one. User must resort to `HAL_HASH_xxx_Accumulate_End_IT()` to enter the last one and retrieve as well the computed digest.
  - c. In DMA mode, multi-buffer HASH and HMAC processing are possible.
    - HASH processing: once initialization is done, MDMAT bit must be set thru `__HAL_HASH_SET_MDMAT()` macro. From that point, each buffer can be fed to the Peripheral thru `HAL_HASH_xxx_Start_DMA()` API. Before entering the last buffer, reset the MDMAT bit with `__HAL_HASH_RESET_MDMAT()` macro then wrap-up the HASH processing in feeding the last input buffer thru the same API `HAL_HASH_xxx_Start_DMA()`. The digest can then be retrieved with a call to API `HAL_HASH_xxx_Finish()`.
    - HMAC processing (requires to resort to extended functions): after initialization, the key and the first input buffer are entered in the Peripheral with the API `HAL_HMACEx_xxx_Step1_2_DMA()`. This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API `HAL_HMACEx_xxx_Step2_DMA()`. At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to `HAL_HMACEx_xxx_Step2_3_DMA()`. The digest can finally be retrieved with a call to API `HAL_HASH_xxx_Finish()`.

6. Context swapping.
  - a. Two APIs are available to suspend HASH or HMAC processing:
    - HAL\_HASH\_SwFeed\_ProcessSuspend() when data are entered by software (polling or IT mode),
    - HAL\_HASH\_DMAFeed\_ProcessSuspend() when data are entered by DMA.
  - b. When HASH or HMAC processing is suspended, HAL\_HASH\_ContextSaving() allows to save in memory the Peripheral context. This context can be restored afterwards to resume the HASH processing thanks to HAL\_HASH\_ContextRestoring().
  - c. Once the HASH Peripheral has been restored to the same configuration as that at suspension time, processing can be restarted with the same API call (same API, same handle, same parameters) as done before the suspension. Relevant parameters to restart at the proper location are internally saved in the HASH handle.
7. Call HAL\_HASH\_DeInit() to deinitialize the HASH peripheral.

### Remarks on message length

1. HAL in interruption mode (interruptions driven)
  - a. Due to HASH peripheral hardware design, the peripheral interruption is triggered every 64 bytes. This is why, for driver implementation simplicity's sake, user is requested to enter a message the length of which is a multiple of 4 bytes.
  - b. When the message length (in bytes) is not a multiple of words, a specific field exists in HASH\_STR to specify which bits to discard at the end of the complete message to process only the message bits and not extra bits.
  - c. If user needs to perform a hash computation of a large input buffer that is spread around various places in memory and where each piece of this input buffer is not necessarily a multiple of 4 bytes in size, it becomes necessary to use a temporary buffer to format the data accordingly before feeding them to the Peripheral. It is advised to the user to
    - achieve the first formatting operation by software then enter the data
    - while the Peripheral is processing the first input set, carry out the second formatting operation by software, to be ready when DINIS occurs.
    - repeat step 2 until the whole message is processed.
1. HAL in DMA mode
  - a. Again, due to hardware design, the DMA transfer to feed the data can only be done on a word-basis. The same field described above in HASH\_STR is used to specify which bits to discard at the end of the DMA transfer to process only the message bits and not extra bits. Due to hardware implementation, this is possible only at the end of the complete message. When several DMA transfers are needed to enter the message, this is not applicable at the end of the intermediary transfers.
  - b. Similarly to the interruption-driven mode, it is suggested to the user to format the consecutive chunks of data by software while the DMA transfer and processing is on-going for the first parts of the message. Due to the 32-bit alignment required for the DMA transfer, it is underlined that the software formatting operation is more complex than in the IT mode.

### Callback registration

1. The compilation define USE\_HAL\_HASH\_REGISTER\_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL\_HASH\_RegisterCallback() to register a user callback.
2. Function @ref HAL\_HASH\_RegisterCallback() allows to register following callbacks: (+) InCpltCallback : callback for input completion. (+) DgstCpltCallback : callback for digest computation completion. (+) ErrorCallback : callback for error. (+) MspInitCallback : HASH MspInit. (+) MspDeInitCallback : HASH MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function @ref HAL\_HASH\_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL\_HASH\_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) InCpltCallback : callback for input completion. (+) DgstCpltCallback : callback for digest computation completion. (+) ErrorCallback : callback for error. (+) MspInitCallback : HASH MspInit. (+) MspDeInitCallback : HASH MspDeInit.

4. By default, after the @ref HAL\_HASH\_Init and if the state is HAL\_HASH\_STATE\_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL\_HASH\_InCpltCallback(), @ref HAL\_HASH\_DgstCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL\_HASH\_Init and @ref HAL\_HASH\_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL\_HASH\_Init and @ref HAL\_HASH\_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\_HASH\_RegisterCallback before calling @ref HAL\_HASH\_DeInit or @ref HAL\_HASH\_Init function. When The compilation define USE\_HAL\_HASH\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

(#) The compilation define USE\_HAL\_HASH\_REGISTER\_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL\_HASH\_RegisterCallback() to register a user callback. (#) Function @ref HAL\_HASH\_RegisterCallback() allows to register following callbacks:

- InCpltCallback : callback for input completion.
- DgstCpltCallback : callback for digest computation completion.
- ErrorCallback : callback for error.
- MspInitCallback : HASH MspInit.
- MspDeInitCallback : HASH 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\_HASH\_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL\_HASH\_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- InCpltCallback : callback for input completion.
- DgstCpltCallback : callback for digest computation completion.
- ErrorCallback : callback for error.
- MspInitCallback : HASH MspInit.
- MspDeInitCallback : HASH MspDeInit. (#) By default, after the @ref HAL\_HASH\_Init and if the state is HAL\_HASH\_STATE\_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL\_HASH\_InCpltCallback(), @ref HAL\_HASH\_DgstCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL\_HASH\_Init and @ref HAL\_HASH\_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL\_HASH\_Init and @ref HAL\_HASH\_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\_HASH\_RegisterCallback before calling @ref HAL\_HASH\_DeInit or @ref HAL\_HASH\_Init function. When The compilation define USE\_HAL\_HASH\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

### 33.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH\_InitTypeDef and create the associated handle
- DeInitialize the HASH peripheral
- Initialize the HASH MCU Specific Package (MSP)
- DeInitialize the HASH MSP

This section provides as well call back functions definitions for user code to manage:

- Input data transfer to Peripheral completion
- Calculated digest retrieval completion
- Error management

This section contains the following APIs:

- ***HAL\_HASH\_Init()***

- *HAL\_HASH\_DeInit()*
- *HAL\_HASH\_MspInit()*
- *HAL\_HASH\_MspDeInit()*
- *HAL\_HASH\_InCpltCallback()*
- *HAL\_HASH\_DgstCpltCallback()*
- *HAL\_HASH\_ErrorCallback()*

### 33.2.3 Polling mode HASH processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- MD5
  - *HAL\_HASH\_MD5\_Start()*
  - *HAL\_HASH\_MD5\_Accmlt()*
  - *HAL\_HASH\_MD5\_Accmlt\_End()*
- SHA1
  - *HAL\_HASH\_SHA1\_Start()*
  - *HAL\_HASH\_SHA1\_Accmlt()*
  - *HAL\_HASH\_SHA1\_Accmlt\_End()*

For a single buffer to be hashed, user can resort to *HAL\_HASH\_xxx\_Start()*.

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to *HAL\_HASH\_xxx\_Accumulate()* and wrap-up the digest computation by a call to *HAL\_HASH\_xxx\_Accumulate\_End()*.

This section contains the following APIs:

- *HAL\_HASH\_MD5\_Start()*
- *HAL\_HASH\_MD5\_Accmlt()*
- *HAL\_HASH\_MD5\_Accmlt\_End()*
- *HAL\_HASH\_SHA1\_Start()*
- *HAL\_HASH\_SHA1\_Accmlt()*
- *HAL\_HASH\_SHA1\_Accmlt\_End()*

### 33.2.4 Interruption mode HASH processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- MD5
  - *HAL\_HASH\_MD5\_Start\_IT()*
  - *HAL\_HASH\_MD5\_Accmlt\_IT()*
  - *HAL\_HASH\_MD5\_Accmlt\_End\_IT()*
- SHA1
  - *HAL\_HASH\_SHA1\_Start\_IT()*
  - *HAL\_HASH\_SHA1\_Accmlt\_IT()*
  - *HAL\_HASH\_SHA1\_Accmlt\_End\_IT()*

API *HAL\_HASH\_IRQHandler()* manages each HASH interruption.

Note that *HAL\_HASH\_IRQHandler()* manages as well HASH Peripheral interruptions when in HMAC processing mode.

This section contains the following APIs:

- *HAL\_HASH\_MD5\_Start\_IT()*
- *HAL\_HASH\_MD5\_Accmlt\_IT()*
- *HAL\_HASH\_MD5\_Accmlt\_End\_IT()*
- *HAL\_HASH\_SHA1\_Start\_IT()*
- *HAL\_HASH\_SHA1\_Accmlt\_IT()*

- [\*HAL\\_HASH\\_SHA1\\_Accmplt\\_End\\_IT\(\)\*](#)
- [\*HAL\\_HASH\\_IRQHandler\(\)\*](#)

### 33.2.5 DMA mode HASH processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- MD5
  - [\*HAL\\_HASH\\_MD5\\_Start\\_DMA\(\)\*](#)
  - [\*HAL\\_HASH\\_MD5\\_Finish\(\)\*](#)
- SHA1
  - [\*HAL\\_HASH\\_SHA1\\_Start\\_DMA\(\)\*](#)
  - [\*HAL\\_HASH\\_SHA1\\_Finish\(\)\*](#)

When resorting to DMA mode to enter the data in the Peripheral, user must resort to [\*HAL\\_HASH\\_xxx\\_Start\\_DMA\(\)\*](#) then read the resulting digest with [\*HAL\\_HASH\\_xxx\\_Finish\(\)\*](#).

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to [\*HAL\\_HASH\\_xxx\\_Start\\_DMA\(\)\*](#). Then, MDMAT bit needs to be reset before the last call to [\*HAL\\_HASH\\_xxx\\_Start\\_DMA\(\)\*](#). Digest is finally retrieved thanks to [\*HAL\\_HASH\\_xxx\\_Finish\(\)\*](#).

This section contains the following APIs:

- [\*HAL\\_HASH\\_MD5\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HASH\\_MD5\\_Finish\(\)\*](#)
- [\*HAL\\_HASH\\_SHA1\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HASH\\_SHA1\\_Finish\(\)\*](#)

### 33.2.6 Polling mode HMAC processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- MD5
  - [\*HAL\\_HMAC\\_MD5\\_Start\(\)\*](#)
- SHA1
  - [\*HAL\\_HMAC\\_SHA1\\_Start\(\)\*](#)

This section contains the following APIs:

- [\*HAL\\_HMAC\\_MD5\\_Start\(\)\*](#)
- [\*HAL\\_HMAC\\_SHA1\\_Start\(\)\*](#)

### 33.2.7 Interrupt mode HMAC processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- MD5
  - [\*HAL\\_HMAC\\_MD5\\_Start\\_IT\(\)\*](#)
- SHA1
  - [\*HAL\\_HMAC\\_SHA1\\_Start\\_IT\(\)\*](#)

This section contains the following APIs:

- [\*HAL\\_HMAC\\_MD5\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_HMAC\\_SHA1\\_Start\\_IT\(\)\*](#)

### 33.2.8 DMA mode HMAC processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- MD5
  - [\*HAL\\_HMAC\\_MD5\\_Start\\_DMA\(\)\*](#)

- SHA1
  - HAL\_HMAC\_SHA1\_Start\_DMA()

When resorting to DMA mode to enter the data in the Peripheral for HMAC processing, user must resort to HAL\_HMAC\_XXX\_Start\_DMA() then read the resulting digest with HAL\_HASH\_XXX\_Finish().

This section contains the following APIs:

- [HAL\\_HMAC\\_MD5\\_Start\\_DMA\(\)](#)
- [HAL\\_HMAC\\_SHA1\\_Start\\_DMA\(\)](#)

### 33.2.9 Peripheral State methods

This section permits to get in run-time the state and the peripheral handle status of the peripheral:

- HAL\_HASH\_GetState()
- HAL\_HASH\_GetStatus()

Additionally, this subsection provides functions allowing to save and restore the HASH or HMAC processing context in case of calculation suspension:

- HAL\_HASH\_ContextSaving()
- HAL\_HASH\_ContextRestoring()

This subsection provides functions allowing to suspend the HASH processing

- when input are fed to the Peripheral by software
  - HAL\_HASH\_SwFeed\_ProcessSuspend()
- when input are fed to the Peripheral by DMA
  - HAL\_HASH\_DMAFeed\_ProcessSuspend()

This section contains the following APIs:

- [HAL\\_HASH\\_GetState\(\)](#)
- [HAL\\_HASH\\_GetStatus\(\)](#)
- [HAL\\_HASH\\_ContextSaving\(\)](#)
- [HAL\\_HASH\\_ContextRestoring\(\)](#)
- [HAL\\_HASH\\_SwFeed\\_ProcessSuspend\(\)](#)
- [HAL\\_HASH\\_DMAFeed\\_ProcessSuspend\(\)](#)
- [HAL\\_HASH\\_GetError\(\)](#)

### 33.2.10 Detailed description of functions

#### HAL\_HASH\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_HASH\_Init (HASH\_HandleTypeDef \* hhash)**

##### Function description

Initialize the HASH according to the specified parameters in the HASH\_HandleTypeDef and create the associated handle.

##### Parameters

- **hhash:** HASH handle

##### Return values

- **HAL:** status

##### Notes

- Only MDMAT and DATATYPE bits of HASH Peripheral are set by HAL\_HASH\_Init(), other configuration bits are set by HASH or HMAC processing APIs.
- MDMAT bit is systematically reset by HAL\_HASH\_Init(). To set it for multi-buffer HASH processing, user needs to resort to \_\_HAL\_HASH\_SET\_MDMAT() macro. For HMAC multi-buffer processing, the relevant APIs manage themselves the MDMAT bit.

### HAL\_HASH\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_HASH\_DeInit (HASH\_HandleTypeDef \* hhash)**

#### Function description

Deinitialize the HASH peripheral.

#### Parameters

- **hhash**: HASH handle.

#### Return values

- **HAL**: status

### HAL\_HASH\_MspInit

#### Function name

**void HAL\_HASH\_MspInit (HASH\_HandleTypeDef \* hhash)**

#### Function description

Initialize the HASH MSP.

#### Parameters

- **hhash**: HASH handle.

#### Return values

- **None**:

### HAL\_HASH\_MspDeInit

#### Function name

**void HAL\_HASH\_MspDeInit (HASH\_HandleTypeDef \* hhash)**

#### Function description

Deinitialize the HASH MSP.

#### Parameters

- **hhash**: HASH handle.

#### Return values

- **None**:

### HAL\_HASH\_InCpltCallback

#### Function name

**void HAL\_HASH\_InCpltCallback (HASH\_HandleTypeDef \* hhash)**

#### Function description

Input data transfer complete call back.

#### Parameters

- **hhash**: HASH handle.

#### Return values

- **None**:

**Notes**

- HAL\_HASH\_InCpltCallback() is called when the complete input message has been fed to the Peripheral. This API is invoked only when input data are entered under interruption or thru DMA.
- In case of HASH or HMAC multi-buffer DMA feeding case (MDMAT bit set), HAL\_HASH\_InCpltCallback() is called at the end of each buffer feeding to the Peripheral.

**HAL\_HASH\_DgstCpltCallback**
**Function name**

```
void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)
```

**Function description**

Digest computation complete call back.

**Parameters**

- **hhash:** HASH handle.

**Return values**

- **None:**

**Notes**

- HAL\_HASH\_DgstCpltCallback() is used under interruption, is not relevant with DMA.

**HAL\_HASH\_ErrorCallback**
**Function name**

```
void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)
```

**Function description**

Error callback.

**Parameters**

- **hhash:** HASH handle.

**Return values**

- **None:**

**Notes**

- Code user can resort to hhash->Status (HAL\_ERROR, HAL\_TIMEOUT,...) to retrieve the error type.

**HAL\_HASH\_SHA1\_Start**
**Function name**

```
HAL_StatusTypeDef HAL_HASH_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
```

**Function description**

Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value

**Return values**

- **HAL:** status



### Notes

- Digest is available in pOutBuffer.

### HAL\_HASH\_MD5\_Start

#### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

#### Function description

Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 16 bytes.
- **Timeout**: Timeout value

#### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.

### HAL\_HASH\_MD5\_Accmlt

#### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Accmlt (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes, must be a multiple of 4.

#### Return values

- **HAL**: status

### Notes

- Consecutive calls to HAL\_HASH\_MD5\_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASH\_MD5\_Accmlt\_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL\_HASH\_MD5\_Accmlt\_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASH\_MD5\_Accmlt\_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

## HAL\_HASH\_SHA1\_Accmlt

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Accmlt (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL:** status

### Notes

- Consecutive calls to HAL\_HASH\_SHA1\_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASH\_SHA1\_Accmlt\_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL\_HASH\_SHA1\_Accmlt\_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASH\_SHA1\_Accmlt\_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

## HAL\_HASH\_MD5\_Accmlt\_End

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Accmlt\_End (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

End computation of a single HASH signature after several calls to HAL\_HASH\_MD5\_Accmlt() API.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

## HAL\_HASH\_SHA1\_Accmlt\_End

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Accmlt\_End (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

End computation of a single HASH signature after several calls to HAL\_HASH\_SHA1\_Accmlt() API.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 20 bytes.
- **Timeout**: Timeout value

### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.

### HAL\_HASH\_SHA1\_Start\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

### Function description

Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest in interruption mode.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 20 bytes.

### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.

### HAL\_HASH\_SHA1\_Accmlt\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Accmlt\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer in interruption mode.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL**: status

**Notes**

- Consecutive calls to HAL\_HASH\_SHA1\_Accmlt\_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASH\_SHA1\_Accmlt\_End\_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASH\_SHA1\_Accmlt\_End\_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

**HAL\_HASH\_SHA1\_Accmlt\_End\_IT**
**Function name**

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Accmlt\_End\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

**Function description**

End computation of a single HASH signature after several calls to HAL\_HASH\_SHA1\_Accmlt\_IT() API.

**Parameters**

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 20 bytes.

**Return values**

- **HAL**: status

**Notes**

- Digest is available in pOutBuffer.

**HAL\_HASH\_MD5\_Start\_IT**
**Function name**

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

**Function description**

Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest in interruption mode.

**Parameters**

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 16 bytes.

**Return values**

- **HAL**: status

**Notes**

- Digest is available in pOutBuffer.

## HAL\_HASH\_MD5\_Accmlt\_IT

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Accmlt\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer in interruption mode.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL:** status

### Notes

- Consecutive calls to HAL\_HASH\_MD5\_Accmlt\_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASH\_MD5\_Accmlt\_End\_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASH\_MD5\_Accmlt\_End\_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

## HAL\_HASH\_MD5\_Accmlt\_End\_IT

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Accmlt\_End\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

### Function description

End computation of a single HASH signature after several calls to HAL\_HASH\_MD5\_Accmlt\_IT() API.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

## HAL\_HASH\_IRQHandler

### Function name

**void HAL\_HASH\_IRQHandler (HASH\_HandleTypeDef \* hhash)**

### Function description

Handle HASH interrupt request.

### Parameters

- **hhash:** HASH handle.

### Return values

- **None:**

### Notes

- HAL\_HASH\_IRQHandler() handles interrupts in HMAC processing as well.
- In case of error reported during the HASH interruption processing, HAL\_HASH\_ErrorCallback() API is called so that user code can manage the error. The error type is available in hhash->Status field.

## HAL\_HASH\_SHA1\_Start\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

Initialize the HASH peripheral in SHA1 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

### Return values

- **HAL:** status

### Notes

- Once the DMA transfer is finished, HAL\_HASH\_SHA1\_Finish() API must be called to retrieve the computed digest.

## HAL\_HASH\_SHA1\_Finish

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Finish (HASH\_HandleTypeDef \* hhash, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

Return the computed digest in SHA1 mode.

### Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value.

### Return values

- **HAL:** status

### Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL\_HASH\_SHA1\_Finish() can be used as well to retrieve the digest in HMAC SHA1 mode.

## HAL\_HASH\_MD5\_Start\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

Initialize the HASH peripheral in MD5 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.

### Return values

- **HAL**: status

### Notes

- Once the DMA transfer is finished, HAL\_HASH\_MD5\_Finish() API must be called to retrieve the computed digest.

## HAL\_HASH\_MD5\_Finish

### Function name

**HAL\_StatusTypeDef HAL\_HASH\_MD5\_Finish (HASH\_HandleTypeDef \* hhash, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

Return the computed digest in MD5 mode.

### Parameters

- **hhash**: HASH handle.
- **pOutBuffer**: pointer to the computed digest. Digest size is 16 bytes.
- **Timeout**: Timeout value.

### Return values

- **HAL**: status

### Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL\_HASH\_MD5\_Finish() can be used as well to retrieve the digest in HMAC MD5 mode.

## HAL\_HMAC\_SHA1\_Start

### Function name

**HAL\_StatusTypeDef HAL\_HMAC\_SHA1\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 20 bytes.
- **Timeout**: Timeout value.

### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

## HAL\_HMAC\_MD5\_Start

### Function name

**HAL\_StatusTypeDef HAL\_HMAC\_MD5\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 16 bytes.
- **Timeout**: Timeout value.

### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

## HAL\_HMAC\_MD5\_Start\_IT

### Function name

**HAL\_StatusTypeDef HAL\_HMAC\_MD5\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

### Function description

Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest in interrupt mode.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 16 bytes.



### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

### HAL\_HMAC\_SHA1\_Start\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HMAC\_SHA1\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

#### Function description

Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest in interrupt mode.

#### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

### HAL\_HMAC\_SHA1\_Start\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMAC\_SHA1\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

Initialize the HASH peripheral in HMAC SHA1 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

#### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

### Return values

- **HAL:** status

**Notes**

- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASH_SHA1_Finish()` API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

**HAL\_HMAC\_MD5\_Start\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMAC\_MD5\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

Initialize the HASH peripheral in HMAC MD5 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

**Return values**

- **HAL:** status

**Notes**

- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASH_MD5_Finish()` API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

**HAL\_HASH\_GetState**
**Function name**

**HAL\_HASH\_StateTypeDef HAL\_HASH\_GetState (HASH\_HandleTypeDef \* hhash)**

**Function description**

Return the HASH handle state.

**Parameters**

- **hhash:** HASH handle.

**Return values**

- **HAL:** HASH state

**Notes**

- The API yields the current state of the handle (BUSY, READY,...).

### HAL\_HASH\_GetStatus

#### Function name

**HAL\_StatusTypeDef HAL\_HASH\_GetStatus (HASH\_HandleTypeDef \* hhash)**

#### Function description

Return the HASH HAL status.

#### Parameters

- **hhash:** HASH handle.

#### Return values

- **HAL:** status

#### Notes

- The API yields the HAL status of the handle: it is the result of the latest HASH processing and allows to report any issue (e.g. HAL\_TIMEOUT).

### HAL\_HASH\_ContextSaving

#### Function name

**void HAL\_HASH\_ContextSaving (HASH\_HandleTypeDef \* hhash, uint8\_t \* pMemBuffer)**

#### Function description

Save the HASH context in case of processing suspension.

#### Parameters

- **hhash:** HASH handle.
- **pMemBuffer:** pointer to the memory buffer where the HASH context is saved.

#### Return values

- **None:**

#### Notes

- The IMR, STR, CR then all the CSR registers are saved in that order. Only the r/w bits are read to be restored later on.
- By default, all the context swap registers (there are HASH\_NUMBER\_OF\_CSR\_REGISTERS of those) are saved.
- pMemBuffer points to a buffer allocated by the user. The buffer size must be at least (HASH\_NUMBER\_OF\_CSR\_REGISTERS + 3) \* 4 uint8 long.

### HAL\_HASH\_ContextRestoring

#### Function name

**void HAL\_HASH\_ContextRestoring (HASH\_HandleTypeDef \* hhash, uint8\_t \* pMemBuffer)**

#### Function description

Restore the HASH context in case of processing resumption.

#### Parameters

- **hhash:** HASH handle.
- **pMemBuffer:** pointer to the memory buffer where the HASH context is stored.

#### Return values

- **None:**

**Notes**

- The IMR, STR, CR then all the CSR registers are restored in that order. Only the r/w bits are restored.
- By default, all the context swap registers (HASH\_NUMBER\_OF\_CSR\_REGISTERS of those) are restored (all of them have been saved by default beforehand).

**HAL\_HASH\_SwFeed\_ProcessSuspend**
**Function name**

```
void HAL_HASH_SwFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)
```

**Function description**

Initiate HASH processing suspension when in polling or interruption mode.

**Parameters**

- **hhash:** HASH handle.

**Return values**

- **None:**

**Notes**

- Set the handle field SuspendRequest to the appropriate value so that the on-going HASH processing is suspended as soon as the required conditions are met. Note that the actual suspension is carried out by the functions HASH\_WriteData() in polling mode and HASH\_IT() in interruption mode.

**HAL\_HASH\_DMAFeed\_ProcessSuspend**
**Function name**

```
HAL_StatusTypeDef HAL_HASH_DMAFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)
```

**Function description**

Suspend the HASH processing when in DMA mode.

**Parameters**

- **hhash:** HASH handle.

**Return values**

- **HAL:** status

**Notes**

- When suspension attempt occurs at the very end of a DMA transfer and all the data have already been entered in the Peripheral, hhash->State is set to HAL\_HASH\_STATE\_READY and the API returns HAL\_ERROR. It is recommended to wrap-up the processing in reading the digest as usual.

**HAL\_HASH\_GetError**
**Function name**

```
uint32_t HAL_HASH_GetError (HASH_HandleTypeDef * hhash)
```

**Function description**

Return the HASH handle error code.

**Parameters**

- **hhash:** pointer to a HASH\_HandleTypeDef structure.

**Return values**

- **HASH:** Error Code

## HASH\_Start

### Function name

**HAL\_StatusTypeDef** HASH\_Start (**HASH\_HandleTypeDef** \* hhash, **uint8\_t** \* pInBuffer, **uint32\_t** Size, **uint8\_t** \* pOutBuffer, **uint32\_t** Timeout, **uint32\_t** Algorithm)

### Function description

Initialize the HASH peripheral, next process pInBuffer then read the computed digest.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest.
- **Timeout**: Timeout value.
- **Algorithm**: HASH algorithm.

### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.

## HASH\_Accumulate

### Function name

**HAL\_StatusTypeDef** HASH\_Accumulate (**HASH\_HandleTypeDef** \* hhash, **uint8\_t** \* pInBuffer, **uint32\_t** Size, **uint32\_t** Algorithm)

### Function description

If not already done, initialize the HASH peripheral then processes pInBuffer.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes, must be a multiple of 4.
- **Algorithm**: HASH algorithm.

### Return values

- **HAL**: status

### Notes

- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

## HASH\_Accumulate\_IT

### Function name

**HAL\_StatusTypeDef** HASH\_Accumulate\_IT (**HASH\_HandleTypeDef** \* hhash, **uint8\_t** \* pInBuffer, **uint32\_t** Size, **uint32\_t** Algorithm)

### Function description

If not already done, initialize the HASH peripheral then processes pInBuffer in interruption mode.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.
- **Algorithm:** HASH algorithm.

### Return values

- **HAL:** status

### Notes

- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

## HASH\_Start\_IT

### Function name

**HAL\_StatusTypeDef HASH\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Algorithm)**

### Function description

Initialize the HASH peripheral, next process pInBuffer then read the computed digest in interruption mode.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Algorithm:** HASH algorithm.

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

## HASH\_Start\_DMA

### Function name

**HAL\_StatusTypeDef HASH\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint32\_t Algorithm)**

### Function description

Initialize the HASH peripheral then initiate a DMA transfer to feed the input buffer to the Peripheral.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **Algorithm:** HASH algorithm.

### Return values

- **HAL:** status

**Notes**

- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

**HASH\_Finish**
**Function name**

**HAL\_StatusTypeDef** HASH\_Finish (HASH\_HandleTypeDef \* hhash, uint8\_t \* pOutBuffer, uint32\_t Timeout)

**Function description**

Return the computed digest.

**Parameters**

- **hhash**: HASH handle.
- **pOutBuffer**: pointer to the computed digest.
- **Timeout**: Timeout value.

**Return values**

- **HAL**: status

**Notes**

- The API waits for DCIS to be set then reads the computed digest.

**HMAC\_Start**
**Function name**

**HAL\_StatusTypeDef** HMAC\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout, uint32\_t Algorithm)

**Function description**

Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest.

**Parameters**

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest.
- **Timeout**: Timeout value.
- **Algorithm**: HASH algorithm.

**Return values**

- **HAL**: status

**Notes**

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

**HMAC\_Start\_IT**
**Function name**

**HAL\_StatusTypeDef** HMAC\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Algorithm)

### Function description

Initialize the HASH peripheral in HMAC mode, next process `plnBuffer` then read the computed digest in interruption mode.

### Parameters

- **hhash**: HASH handle.
- **plnBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest.
- **Algorithm**: HASH algorithm.

### Return values

- **HAL**: status

### Notes

- Digest is available in `pOutBuffer`.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.

## HMAC\_Start\_DMA

### Function name

**HAL\_StatusTypeDef HMAC\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* plnBuffer, uint32\_t Size, uint32\_t Algorithm)**

### Function description

Initialize the HASH peripheral in HMAC mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

### Parameters

- **hhash**: HASH handle.
- **plnBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **Algorithm**: HASH algorithm.

### Return values

- **HAL**: status

### Notes

- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- In case of multi-buffer HMAC processing, the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only the length of the last buffer of the thread doesn't have to be a multiple of 4.

## 33.3 HASH Firmware driver defines

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

### 33.3.1 HASH

HASH

*HASH algorithm mode*

#### HASH\_ALGOMODE\_HASH

Algorithm is HASH



#### **HASH\_ALGOMODE\_HMAC**

Algorithm is HMAC  
**HASH algorithm selection**

#### **HASH\_ALGOSELECTION\_SHA1**

HASH function is SHA1

#### **HASH\_ALGOSELECTION\_MD5**

HASH function is MD5

#### **HASH\_ALGOSELECTION\_SHA224**

HASH function is SHA224

#### **HASH\_ALGOSELECTION\_SHA256**

HASH function is SHA256  
**HASH API alias**

#### **HAL\_HASHEx\_IRQHandler**

is re-directed to  
**HASH input data type**

#### **HASH\_DATATYPE\_32B**

32-bit data. No swapping

#### **HASH\_DATATYPE\_16B**

16-bit data. Each half word is swapped

#### **HASH\_DATATYPE\_8B**

8-bit data. All bytes are swapped

#### **HASH\_DATATYPE\_1B**

1-bit data. In the word all bits are swapped  
**HASH Digest Calculation Status**

#### **HASH\_DIGEST\_CALCULATION\_NOT\_STARTED**

DCAL not set after input data written in DIN register

#### **HASH\_DIGEST\_CALCULATION\_STARTED**

DCAL set after input data written in DIN register  
**HASH DMA suspension words limit**

#### **HASH\_DMA\_SUSPENSION\_WORDS\_LIMIT**

Number of words below which DMA suspension is aborted  
**HASH Error Definition**

#### **HAL\_HASH\_ERROR\_NONE**

No error

#### **HAL\_HASH\_ERROR\_IT**

IT-based process error

#### **HAL\_HASH\_ERROR\_DMA**

DMA-based process error  
**HASH Exported Macros**

### **\_\_HAL\_HASH\_GET\_FLAG**

**Description:**

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

**Parameters:**

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `HASH_FLAG_DINIS` A new block can be entered into the input buffer.
  - `HASH_FLAG_DCIS` Digest calculation complete.
  - `HASH_FLAG_DMAS` DMA interface is enabled (`DMAE=1`) or a transfer is ongoing.
  - `HASH_FLAG_BUSY` The hash core is Busy : processing a block of data.
  - `HASH_FLAG_DINNE` DIN not empty : the input buffer contains at least one word of data.

**Return value:**

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

### **\_\_HAL\_HASH\_CLEAR\_FLAG**

**Description:**

- Clear the specified HASH flag.

**Parameters:**

- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
  - `HASH_FLAG_DINIS` A new block can be entered into the input buffer.
  - `HASH_FLAG_DCIS` Digest calculation complete

**Return value:**

- None

### **\_\_HAL\_HASH\_ENABLE\_IT**

**Description:**

- Enable the specified HASH interrupt.

**Parameters:**

- `__INTERRUPT__`: specifies the HASH interrupt source to enable. This parameter can be one of the following values:
  - `HASH_IT_DINI` A new block can be entered into the input buffer (DIN)
  - `HASH_IT_DCI` Digest calculation complete

**Return value:**

- None

### **\_\_HAL\_HASH\_DISABLE\_IT**

**Description:**

- Disable the specified HASH interrupt.

**Parameters:**

- `__INTERRUPT__`: specifies the HASH interrupt source to disable. This parameter can be one of the following values:
  - `HASH_IT_DINI` A new block can be entered into the input buffer (DIN)
  - `HASH_IT_DCI` Digest calculation complete

**Return value:**

- None

### **\_\_HAL\_HASH\_RESET\_HANDLE\_STATE**

**Description:**

- Reset HASH handle state.

**Parameters:**

- `__HANDLE__`: HASH handle.

**Return value:**

- None

### **\_\_HAL\_HASH\_RESET\_HANDLE\_STATUS**

**Description:**

- Reset HASH handle status.

**Parameters:**

- `__HANDLE__`: HASH handle.

**Return value:**

- None

### **\_\_HAL\_HASH\_SET\_MDMAT**

**Description:**

- Enable the multi-buffer DMA transfer mode.

**Return value:**

- None

**Notes:**

- This bit is set when hashing large files when multiple DMA transfers are needed.

### **\_\_HAL\_HASH\_RESET\_MDMAT**

**Description:**

- Disable the multi-buffer DMA transfer mode.

**Return value:**

- None

### **\_\_HAL\_HASH\_START\_DIGEST**

**Description:**

- Start the digest computation.

**Return value:**

- None

### **\_\_HAL\_HASH\_SET\_NBVALIDBITS**

**Description:**

- Set the number of valid bits in the last word written in data register DIN.

**Parameters:**

- `__SIZE__`: size in bytes of last data written in Data register.

**Return value:**

- None

### **\_\_HAL\_HASH\_INIT**

**Description:**

- Reset the HASH core.

**Return value:**

- None

***HASH flags definitions*****HASH\_FLAG\_DINIS**

16 locations are free in the DIN : a new block can be entered in the Peripheral

**HASH\_FLAG\_DCIS**

Digest calculation complete

**HASH\_FLAG\_DMAS**

DMA interface is enabled (DMAE=1) or a transfer is ongoing

**HASH\_FLAG\_BUSY**

The hash core is Busy, processing a block of data

**HASH\_FLAG\_DINNE**

DIN not empty : the input buffer contains at least one word of data

***HMAC key length type*****HASH\_HMAC\_KEYTYPE\_SHORTKEY**

HMAC Key size is  $\leq 64$  bytes

**HASH\_HMAC\_KEYTYPE\_LONGKEY**

HMAC Key size is  $> 64$  bytes

***HASH interrupts definitions*****HASH\_IT\_DINI**

A new block can be entered into the input buffer (DIN)

**HASH\_IT\_DCI**

Digest calculation complete

***HASH Number of Context Swap Registers*****HASH\_NUMBER\_OF\_CSR\_REGISTERS**

Number of Context Swap Registers

***HASH TimeOut Value*****HASH\_TIMEOUTVALUE**

Time-out value

## 34 HAL HASH Extension Driver

### 34.1 HASHEx Firmware driver API description

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

#### 34.1.1 HASH peripheral extended features

The SHA-224 and SHA-256 HASH and HMAC processing can be carried out exactly the same way as for SHA-1 or MD-5 algorithms.

1. Three modes are available.
  - a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL\_HASHEx\_xxx\_Start()
  - b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL\_HASHEx\_xxx\_Start\_IT()
  - c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL\_HASHEx\_xxx\_Start\_DMA(). Note that in DMA mode, a call to HAL\_HASHEx\_xxx\_Finish() is then required to retrieve the digest.
2. Multi-buffer processing is possible in polling, interrupt and DMA modes.
  - a. In polling mode, only multi-buffer HASH processing is possible. API HAL\_HASHEx\_xxx\_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL\_HASHEx\_xxx\_Accumulate\_End() to enter the last one and retrieve as well the computed digest.
  - b. In interrupt mode, API HAL\_HASHEx\_xxx\_Accumulate\_IT() must be called for each input buffer, except for the last one. User must resort to HAL\_HASHEx\_xxx\_Accumulate\_End\_IT() to enter the last one and retrieve as well the computed digest.
  - c. In DMA mode, multi-buffer HASH and HMAC processing are possible.
    - HASH processing: once initialization is done, MDMAT bit must be set thru \_\_HAL\_HASH\_SET\_MDMAT() macro. From that point, each buffer can be fed to the Peripheral thru HAL\_HASHEx\_xxx\_Start\_DMA() API. Before entering the last buffer, reset the MDMAT bit with \_\_HAL\_HASH\_RESET\_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer thru the same API HAL\_HASHEx\_xxx\_Start\_DMA(). The digest can then be retrieved with a call to API HAL\_HASHEx\_xxx\_Finish().
    - HMAC processing (MD-5, SHA-1, SHA-224 and SHA-256 must all resort to extended functions): after initialization, the key and the first input buffer are entered in the Peripheral with the API HAL\_HMACEx\_xxx\_Step1\_2\_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL\_HMACEx\_xxx\_Step2\_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL\_HMACEx\_xxx\_Step2\_3\_DMA(). The digest can finally be retrieved with a call to API HAL\_HASH\_xxx\_Finish() for MD-5 and SHA-1, to HAL\_HASHEx\_xxx\_Finish() for SHA-224 and SHA-256.

#### 34.1.2 Polling mode HASH extended processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
  - HAL\_HASHEx\_SHA224\_Start()
  - HAL\_HASHEx\_SHA224\_Accmlt()
  - HAL\_HASHEx\_SHA224\_Accmlt\_End()
- SHA256
  - HAL\_HASHEx\_SHA256\_Start()
  - HAL\_HASHEx\_SHA256\_Accmlt()
  - HAL\_HASHEx\_SHA256\_Accmlt\_End()

For a single buffer to be hashed, user can resort to HAL\_HASH\_xxx\_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to `HAL_HASHEx_xxx_Accumulate()` and wrap-up the digest computation by a call to `HAL_HASHEx_xxx_Accumulate_End()`.

This section contains the following APIs:

- `HAL_HASHEx_SHA224_Start()`
- `HAL_HASHEx_SHA224_Accmlt()`
- `HAL_HASHEx_SHA224_Accmlt_End()`
- `HAL_HASHEx_SHA256_Start()`
- `HAL_HASHEx_SHA256_Accmlt()`
- `HAL_HASHEx_SHA256_Accmlt_End()`

### 34.1.3 Interruption mode HASH extended processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
  - `HAL_HASHEx_SHA224_Start_IT()`
  - `HAL_HASHEx_SHA224_Accmlt_IT()`
  - `HAL_HASHEx_SHA224_Accmlt_End_IT()`
- SHA256
  - `HAL_HASHEx_SHA256_Start_IT()`
  - `HAL_HASHEx_SHA256_Accmlt_IT()`
  - `HAL_HASHEx_SHA256_Accmlt_End_IT()`

This section contains the following APIs:

- `HAL_HASHEx_SHA224_Start_IT()`
- `HAL_HASHEx_SHA224_Accmlt_IT()`
- `HAL_HASHEx_SHA224_Accmlt_End_IT()`
- `HAL_HASHEx_SHA256_Start_IT()`
- `HAL_HASHEx_SHA256_Accmlt_IT()`
- `HAL_HASHEx_SHA256_Accmlt_End_IT()`

### 34.1.4 DMA mode HASH extended processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
  - `HAL_HASHEx_SHA224_Start_DMA()`
  - `HAL_HASHEx_SHA224_Finish()`
- SHA256
  - `HAL_HASHEx_SHA256_Start_DMA()`
  - `HAL_HASHEx_SHA256_Finish()`

When resorting to DMA mode to enter the data in the Peripheral, user must resort to `HAL_HASHEx_xxx_Start_DMA()` then read the resulting digest with `HAL_HASHEx_xxx_Finish()`.

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to `HAL_HASHEx_xxx_Start_DMA()`. Then, MDMAT bit needs to be reset before the last call to `HAL_HASHEx_xxx_Start_DMA()`. Digest is finally retrieved thanks to `HAL_HASHEx_xxx_Finish()`.

This section contains the following APIs:

- `HAL_HASHEx_SHA224_Start_DMA()`
- `HAL_HASHEx_SHA224_Finish()`
- `HAL_HASHEx_SHA256_Start_DMA()`
- `HAL_HASHEx_SHA256_Finish()`

### 34.1.5 Polling mode HMAC extended processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL\_HMACEx\_SHA224\_Start()
- SHA256
  - HAL\_HMACEx\_SHA256\_Start()

This section contains the following APIs:

- [\*HAL\\_HMACEx\\_SHA224\\_Start\(\)\*](#)
- [\*HAL\\_HMACEx\\_SHA256\\_Start\(\)\*](#)

### 34.1.6 Interrupt mode HMAC extended processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL\_HMACEx\_SHA224\_Start\_IT()
- SHA256
  - HAL\_HMACEx\_SHA256\_Start\_IT()

This section contains the following APIs:

- [\*HAL\\_HMACEx\\_SHA224\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_HMACEx\\_SHA256\\_Start\\_IT\(\)\*](#)

### 34.1.7 DMA mode HMAC extended processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL\_HMACEx\_SHA224\_Start\_DMA()
- SHA256
  - HAL\_HMACEx\_SHA256\_Start\_DMA()

When resorting to DMA mode to enter the data in the Peripheral for HMAC processing, user must resort to HAL\_HMACEx\_xxx\_Start\_DMA() then read the resulting digest with HAL\_HASHEx\_xxx\_Finish().

This section contains the following APIs:

- [\*HAL\\_HMACEx\\_SHA224\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HMACEx\\_SHA256\\_Start\\_DMA\(\)\*](#)

### 34.1.8 Multi-buffer DMA mode HMAC extended processing functions

This section provides functions to manage HMAC multi-buffer DMA-based processing for MD5, SHA1, SHA224 and SHA256 algorithms.

- MD5
  - HAL\_HMACEx\_MD5\_Step1\_2\_DMA()
  - HAL\_HMACEx\_MD5\_Step2\_DMA()
  - HAL\_HMACEx\_MD5\_Step2\_3\_DMA()
- SHA1
  - HAL\_HMACEx\_SHA1\_Step1\_2\_DMA()
  - HAL\_HMACEx\_SHA1\_Step2\_DMA()
  - HAL\_HMACEx\_SHA1\_Step2\_3\_DMA()
- SHA256
  - HAL\_HMACEx\_SHA224\_Step1\_2\_DMA()
  - HAL\_HMACEx\_SHA224\_Step2\_DMA()
  - HAL\_HMACEx\_SHA224\_Step2\_3\_DMA()

- SHA256
  - HAL\_HMACEx\_SHA256\_Step1\_2\_DMA()
  - HAL\_HMACEx\_SHA256\_Step2\_DMA()
  - HAL\_HMACEx\_SHA256\_Step2\_3\_DMA()

User must first start-up the multi-buffer DMA-based HMAC computation in calling HAL\_HMACEx\_xxx\_Step1\_2\_DMA(). This carries out HMAC step 1 and initiates step 2 with the first input buffer. The following buffers are next fed to the Peripheral with a call to the API HAL\_HMACEx\_xxx\_Step2\_DMA(). There may be several consecutive calls to this API.

Multi-buffer DMA-based HMAC computation is wrapped up by a call to HAL\_HMACEx\_xxx\_Step2\_3\_DMA(). This finishes step 2 in feeding the last input buffer to the Peripheral then carries out step 3.

Digest is retrieved by a call to HAL\_HASH\_xxx\_Finish() for MD-5 or SHA-1, to HAL\_HASHEX\_xxx\_Finish() for SHA-224 or SHA-256.

If only two buffers need to be consecutively processed, a call to HAL\_HMACEx\_xxx\_Step1\_2\_DMA() followed by a call to HAL\_HMACEx\_xxx\_Step2\_3\_DMA() is sufficient.

This section contains the following APIs:

- [HAL\\_HMACEx\\_MD5\\_Step1\\_2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_MD5\\_Step2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_MD5\\_Step2\\_3\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA1\\_Step1\\_2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA1\\_Step2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA1\\_Step2\\_3\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA224\\_Step1\\_2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA224\\_Step2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA224\\_Step2\\_3\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA256\\_Step1\\_2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA256\\_Step2\\_DMA\(\)](#)
- [HAL\\_HMACEx\\_SHA256\\_Step2\\_3\\_DMA\(\)](#)

### 34.1.9 Detailed description of functions

#### HAL\_HASHEX\_SHA224\_Start

##### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

##### Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest.

##### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

##### Return values

- **HAL:** status

##### Notes

- Digest is available in pOutBuffer.



## HAL\_HASHEX\_SHA224\_Accmlt

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Accmlt (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL:** status

### Notes

- Consecutive calls to HAL\_HASHEX\_SHA224\_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASHEX\_SHA224\_Accmlt\_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL\_HASHEX\_SHA224\_Accmlt\_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASHEX\_SHA224\_Accmlt\_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

## HAL\_HASHEX\_SHA224\_Accmlt\_End

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Accmlt\_End (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

End computation of a single HASH signature after several calls to HAL\_HASHEX\_SHA224\_Accmlt() API.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

## HAL\_HASHEX\_SHA256\_Start

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

### HAL\_HASHEX\_SHA256\_Accmlt

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Accmlt (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL:** status

### Notes

- Consecutive calls to HAL\_HASHEX\_SHA256\_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASHEX\_SHA256\_Accmlt\_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL\_HASHEX\_SHA256\_Accmlt\_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASHEX\_SHA256\_Accmlt\_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

### HAL\_HASHEX\_SHA256\_Accmlt\_End

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Accmlt\_End (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

### Function description

End computation of a single HASH signature after several calls to HAL\_HASHEX\_SHA256\_Accmlt() API.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

### HAL\_HASHEX\_SHA224\_Start\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

#### Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest in interruption mode.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

### Return values

- **HAL:** status

### Notes

- Digest is available in pOutBuffer.

### HAL\_HASHEX\_SHA224\_AccmIt\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_AccmIt\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer in interruption mode.

### Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL:** status

## Notes

- Consecutive calls to HAL\_HASHEX\_SHA224\_Accmlt\_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASHEX\_SHA224\_Accmlt\_End\_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASHEX\_SHA224\_Accmlt\_End\_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

### HAL\_HASHEX\_SHA224\_Accmlt\_End\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Accmlt\_End\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

#### Function description

End computation of a single HASH signature after several calls to HAL\_HASHEX\_SHA224\_Accmlt\_IT() API.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 28 bytes.

#### Return values

- **HAL**: status

## Notes

- Digest is available in pOutBuffer.

### HAL\_HASHEX\_SHA256\_Start\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

#### Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest in interruption mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 32 bytes.

#### Return values

- **HAL**: status

## Notes

- Digest is available in pOutBuffer.

## HAL\_HASHEX\_SHA256\_Accmlt\_IT

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Accmlt\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer in interruption mode.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes, must be a multiple of 4.

### Return values

- **HAL**: status

### Notes

- Consecutive calls to HAL\_HASHEX\_SHA256\_Accmlt\_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL\_HASHEX\_SHA256\_Accmlt\_End\_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL\_HASHEX\_SHA256\_Accmlt\_End\_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

## HAL\_HASHEX\_SHA256\_Accmlt\_End\_IT

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Accmlt\_End\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

### Function description

End computation of a single HASH signature after several calls to HAL\_HASHEX\_SHA256\_Accmlt\_IT() API.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 32 bytes.

### Return values

- **HAL**: status

### Notes

- Digest is available in pOutBuffer.

## HAL\_HASHEX\_SHA224\_Start\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

### Function description

Initialize the HASH peripheral in SHA224 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.

### Return values

- **HAL**: status

### Notes

- Once the DMA transfer is finished, HAL\_HASHEX\_SHA224\_Finish() API must be called to retrieve the computed digest.

### HAL\_HASHEX\_SHA224\_Finish

#### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA224\_Finish (HASH\_HandleTypeDef \* hhash, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

#### Function description

Return the computed digest in SHA224 mode.

#### Parameters

- **hhash**: HASH handle.
- **pOutBuffer**: pointer to the computed digest. Digest size is 28 bytes.
- **Timeout**: Timeout value.

#### Return values

- **HAL**: status

#### Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL\_HASHEX\_SHA224\_Finish() can be used as well to retrieve the digest in HMAC SHA224 mode.

### HAL\_HASHEX\_SHA256\_Start\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

Initialize the HASH peripheral in SHA256 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

**Notes**

- Once the DMA transfer is finished, HAL\_HASHEX\_SHA256\_Finish() API must be called to retrieve the computed digest.

**HAL\_HASHEX\_SHA256\_Finish**
**Function name**

**HAL\_StatusTypeDef HAL\_HASHEX\_SHA256\_Finish (HASH\_HandleTypeDef \* hhash, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

**Function description**

Return the computed digest in SHA256 mode.

**Parameters**

- **hhash**: HASH handle.
- **pOutBuffer**: pointer to the computed digest. Digest size is 32 bytes.
- **Timeout**: Timeout value.

**Return values**

- **HAL**: status

**Notes**

- The API waits for DCIS to be set then reads the computed digest.
- HAL\_HASHEX\_SHA256\_Finish() can be used as well to retrieve the digest in HMAC SHA256 mode.

**HAL\_HMACEx\_SHA224\_Start**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

**Function description**

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest.

**Parameters**

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (buffer to be hashed).
- **Size**: length of the input buffer in bytes.
- **pOutBuffer**: pointer to the computed digest. Digest size is 28 bytes.
- **Timeout**: Timeout value.

**Return values**

- **HAL**: status

**Notes**

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

**HAL\_HMACEx\_SHA256\_Start**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**

**Function description**

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

**Return values**

- **HAL:** status

**Notes**

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

**HAL\_HMACEx\_SHA224\_Start\_IT**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

**Function description**

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest in interrupt mode.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

**Return values**

- **HAL:** status

**Notes**

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

**HAL\_HMACEx\_SHA256\_Start\_IT**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**

**Function description**

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest in interrupt mode.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.



**Return values**

- **HAL:** status

**Notes**

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

**HAL\_HMACEx\_SHA224\_Start\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

**Return values**

- **HAL:** status

**Notes**

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL\_HASH\_STATE\_READY), HAL\_HASHEx\_SHA224\_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

**HAL\_HMACEx\_SHA256\_Start\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

**Return values**

- **HAL:** status

**Notes**

- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASHEX_SHA256_Finish()` API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

**HAL\_HMACEx\_MD5\_Step1\_2\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_MD5\_Step1\_2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

MD5 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

**Return values**

- **HAL:** status

**Notes**

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

**HAL\_HMACEx\_MD5\_Step2\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_MD5\_Step2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

MD5 HMAC step 2 in multi-buffer DMA mode.

**Parameters**

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

**Return values**

- **HAL:** status

## Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

### HAL\_HMACEx\_MD5\_Step2\_3\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_MD5\_Step2\_3\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

MD5 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

## Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASHEX_SHA256_Finish()` API must be called to retrieve the computed digest.

### HAL\_HMACEx\_SHA1\_Step1\_2\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA1\_Step1\_2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA1 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

**Notes**

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

**HAL\_HMACEx\_SHA1\_Step2\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA1\_Step2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

SHA1 HMAC step 2 in multi-buffer DMA mode.

**Parameters**

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

**Return values**

- **HAL**: status

**Notes**

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

**HAL\_HMACEx\_SHA1\_Step2\_3\_DMA**
**Function name**

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA1\_Step2\_3\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

**Function description**

SHA1 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

**Parameters**

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

**Return values**

- **HAL**: status

## Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASHEx_SHA256_Finish()` API must be called to retrieve the computed digest.

### HAL\_HMACEx\_SHA224\_Step1\_2\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Step1\_2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA224 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

## Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

### HAL\_HMACEx\_SHA224\_Step2\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Step2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA224 HMAC step 2 in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

## Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

### HAL\_HMACEx\_SHA224\_Step2\_3\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Step2\_3\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA224 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

## Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASHEX_SHA256_Finish()` API must be called to retrieve the computed digest.

### HAL\_HMACEx\_SHA256\_Step1\_2\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Step1\_2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA256 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

## Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

### HAL\_HMACEx\_SHA256\_Step2\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Step2\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA256 HMAC step 2 in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

## Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

### HAL\_HMACEx\_SHA256\_Step2\_3\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Step2\_3\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**

#### Function description

SHA256 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

#### Parameters

- **hhash**: HASH handle.
- **pInBuffer**: pointer to the input buffer (message buffer).
- **Size**: length of the input buffer in bytes.

#### Return values

- **HAL**: status

### Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- Once the DMA transfers are finished (indicated by `hhash->State` set back to `HAL_HASH_STATE_READY`), `HAL_HASHEX_SHA256_Finish()` API must be called to retrieve the computed digest.



## 35 HAL HCD Generic Driver

### 35.1 HCD Firmware driver registers structures

#### 35.1.1 HCD\_HandleTypeDef

*HCD\_HandleTypeDef* is defined in the `stm32f4xx_hal_hcd.h`

##### Data Fields

- *HCD\_TypeDef \* Instance*
- *HCD\_InitTypeDef Init*
- *HCD\_HCTypeDef hc*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HCD\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*
- *void \* pData*

##### Field Documentation

- *HCD\_TypeDef\* HCD\_HandleTypeDef::Instance*  
Register base address
- *HCD\_InitTypeDef HCD\_HandleTypeDef::Init*  
HCD required parameters
- *HCD\_HCTypeDef HCD\_HandleTypeDef::hc[16]*  
Host channels parameters
- *HAL\_LockTypeDef HCD\_HandleTypeDef::Lock*  
HCD peripheral status
- *\_\_IO HCD\_StateTypeDef HCD\_HandleTypeDef::State*  
HCD communication state
- *\_\_IO uint32\_t HCD\_HandleTypeDef::ErrorCode*  
HCD Error code
- *void\* HCD\_HandleTypeDef::pData*  
Pointer Stack Handler

### 35.2 HCD Firmware driver API description

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

#### 35.2.1 How to use this driver

1. Declare a `HCD_HandleTypeDef` handle structure, for example: `HCD_HandleTypeDef hhcd;`
2. Fill parameters of `Init` structure in HCD handle
3. Call `HAL_HCD_Init()` API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the `HAL_HCD_MspInit()` API:
  - a. Enable the HCD/USB Low Level interface clock using the following macros
    - `__HAL_RCC_USB_OTG_FS_CLK_ENABLE();`
    - `__HAL_RCC_USB_OTG_HS_CLK_ENABLE();` (For High Speed Mode)
    - `__HAL_RCC_USB_OTG_HS_ULPI_CLK_ENABLE();` (For High Speed Mode)
  - b. Initialize the related GPIO clocks
  - c. Configure HCD pin-out
  - d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
  - a. `hhcd.pData = phost;`

6. Enable HCD transmission and reception:
  - a. `HAL_HCD_Start();`

### 35.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- `HAL_HCD_Init()`
- `HAL_HCD_HC_Init()`
- `HAL_HCD_HC_Halt()`
- `HAL_HCD_DeInit()`
- `HAL_HCD_Msplnit()`
- `HAL_HCD_MspDeInit()`

### 35.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USB Host Data Transfer

This section contains the following APIs:

- `HAL_HCD_HC_SubmitRequest()`
- `HAL_HCD_IRQHandler()`
- `HAL_HCD_SOF_Callback()`
- `HAL_HCD_Connect_Callback()`
- `HAL_HCD_Disconnect_Callback()`
- `HAL_HCD_PortEnabled_Callback()`
- `HAL_HCD_PortDisabled_Callback()`
- `HAL_HCD_HC_NotifyURBChange_Callback()`

### 35.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

This section contains the following APIs:

- `HAL_HCD_Start()`
- `HAL_HCD_Stop()`
- `HAL_HCD_ResetPort()`

### 35.2.5 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_HCD_GetState()`
- `HAL_HCD_HC_GetURBState()`
- `HAL_HCD_HC_GetXferCount()`
- `HAL_HCD_HC_GetState()`
- `HAL_HCD_GetCurrentFrame()`
- `HAL_HCD_GetCurrentSpeed()`

### 35.2.6 Detailed description of functions

#### HAL\_HCD\_Init

##### Function name

`HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)`

##### Function description

Initialize the host driver.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **HAL**: status

**HAL\_HCD\_Delnit**
**Function name**

```
HAL_StatusTypeDef HAL_HCD_Delnit (HCD_HandleTypeDef * hhcd)
```

**Function description**

Deinitialize the host driver.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **HAL**: status

**HAL\_HCD\_HC\_Init**
**Function name**

```
HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum,
uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)
```

**Function description**

Initialize a host channel.

**Parameters**

- **hhcd**: HCD handle
- **ch\_num**: Channel number. This parameter can be a value from 1 to 15
- **epnum**: Endpoint number. This parameter can be a value from 1 to 15
- **dev\_address**: Current device address This parameter can be a value from 0 to 255
- **speed**: Current device speed. This parameter can be one of these values: HCD\_SPEED\_HIGH: High speed mode, HCD\_SPEED\_FULL: Full speed mode, HCD\_SPEED\_LOW: Low speed mode
- **ep\_type**: Endpoint Type. This parameter can be one of these values: EP\_TYPE\_CTRL: Control type, EP\_TYPE\_ISOC: Isochronous type, EP\_TYPE\_BULK: Bulk type, EP\_TYPE\_INTR: Interrupt type
- **mps**: Max Packet Size. This parameter can be a value from 0 to 32K

**Return values**

- **HAL**: status

**HAL\_HCD\_HC\_Halt**
**Function name**

```
HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)
```

**Function description**

Halt a host channel.

**Parameters**

- **hhcd**: HCD handle
- **ch\_num**: Channel number. This parameter can be a value from 1 to 15

**Return values**

- **HAL**: status

### HAL\_HCD\_Msplnit

#### Function name

```
void HAL_HCD_Msplnit (HCD_HandleTypeDef * hhcd)
```

#### Function description

Initialize the HCD MSP.

#### Parameters

- **hhcd**: HCD handle

#### Return values

- **None**:

### HAL\_HCD\_MspDeInit

#### Function name

```
void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)
```

#### Function description

Deinitialize the HCD MSP.

#### Parameters

- **hhcd**: HCD handle

#### Return values

- **None**:

### HAL\_HCD\_HC\_SubmitRequest

#### Function name

```
HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num,
uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)
```

#### Function description

Submit a new URB for processing.

#### Parameters

- **hhcd**: HCD handle
- **ch\_num**: Channel number. This parameter can be a value from 1 to 15
- **direction**: Channel number. This parameter can be one of these values: 0 : Output / 1 : Input
- **ep\_type**: Endpoint Type. This parameter can be one of these values: EP\_TYPE\_CTRL: Control type/ EP\_TYPE\_ISOC: Isochronous type/ EP\_TYPE\_BULK: Bulk type/ EP\_TYPE\_INTR: Interrupt type/
- **token**: Endpoint Type. This parameter can be one of these values: 0: HC\_PID\_SETUP / 1: HC\_PID\_DATA1
- **pbuff**: pointer to URB data
- **length**: Length of URB data
- **do\_ping**: activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active

#### Return values

- **HAL**: status

### HAL\_HCD\_IRQHandler

#### Function name

```
void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)
```

### Function description

Handle HCD interrupt request.

### Parameters

- **hhcd**: HCD handle

### Return values

- **None**:

**HAL\_HCD\_SOF\_Callback**

### Function name

**void HAL\_HCD\_SOF\_Callback (HCD\_HandleTypeDef \* hhcd)**

### Function description

SOF callback.

### Parameters

- **hhcd**: HCD handle

### Return values

- **None**:

**HAL\_HCD\_Connect\_Callback**

### Function name

**void HAL\_HCD\_Connect\_Callback (HCD\_HandleTypeDef \* hhcd)**

### Function description

Connection Event callback.

### Parameters

- **hhcd**: HCD handle

### Return values

- **None**:

**HAL\_HCD\_Disconnect\_Callback**

### Function name

**void HAL\_HCD\_Disconnect\_Callback (HCD\_HandleTypeDef \* hhcd)**

### Function description

Disconnection Event callback.

### Parameters

- **hhcd**: HCD handle

### Return values

- **None**:

**HAL\_HCD\_PortEnabled\_Callback**

### Function name

**void HAL\_HCD\_PortEnabled\_Callback (HCD\_HandleTypeDef \* hhcd)**

### Function description

Port Enabled Event callback.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **None**:

**HAL\_HCD\_PortDisabled\_Callback**
**Function name**
**void HAL\_HCD\_PortDisabled\_Callback (HCD\_HandleTypeDef \* hhcd)**
**Function description**

Port Disabled Event callback.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **None**:

**HAL\_HCD\_HC\_NotifyURBChange\_Callback**
**Function name**
**void HAL\_HCD\_HC\_NotifyURBChange\_Callback (HCD\_HandleTypeDef \* hhcd, uint8\_t chnum, HCD\_URBStateTypeDef urb\_state)**
**Function description**

Notify URB state change callback.

**Parameters**

- **hhcd**: HCD handle
- **chnum**: Channel number. This parameter can be a value from 1 to 15
- **urb\_state**: This parameter can be one of these values: URB\_IDLE/ URB\_DONE/ URB\_NOTREADY/ URB\_NYET/ URB\_ERROR/ URB\_STALL/

**Return values**

- **None**:

**HAL\_HCD\_ResetPort**
**Function name**
**HAL\_StatusTypeDef HAL\_HCD\_ResetPort (HCD\_HandleTypeDef \* hhcd)**
**Function description**

Reset the host port.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **HAL**: status

**HAL\_HCD\_Start**
**Function name**
**HAL\_StatusTypeDef HAL\_HCD\_Start (HCD\_HandleTypeDef \* hhcd)**

**Function description**

Start the host driver.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **HAL**: status

**HAL\_HCD\_Stop**
**Function name**

**HAL\_StatusTypeDef HAL\_HCD\_Stop (HCD\_HandleTypeDef \* hhcd)**

**Function description**

Stop the host driver.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **HAL**: status

**HAL\_HCD\_GetState**
**Function name**

**HCD\_StateTypeDef HAL\_HCD\_GetState (HCD\_HandleTypeDef \* hhcd)**

**Function description**

Return the HCD handle state.

**Parameters**

- **hhcd**: HCD handle

**Return values**

- **HAL**: state

**HAL\_HCD\_HC\_GetURBState**
**Function name**

**HCD\_URBStateTypeDef HAL\_HCD\_HC\_GetURBState (HCD\_HandleTypeDef \* hhcd, uint8\_t chnum)**

**Function description**

Return URB state for a channel.

**Parameters**

- **hhcd**: HCD handle
- **chnum**: Channel number. This parameter can be a value from 1 to 15

**Return values**

- **URB**: state. This parameter can be one of these values: URB\_IDLE/ URB\_DONE/ URB\_NOTREADY/ URB\_NYET/ URB\_ERROR/ URB\_STALL

**HAL\_HCD\_HC\_GetState**
**Function name**

**HCD\_HCStateTypeDef HAL\_HCD\_HC\_GetState (HCD\_HandleTypeDef \* hhcd, uint8\_t chnum)**

### Function description

Return the Host Channel state.

### Parameters

- **hhcd**: HCD handle
- **chnum**: Channel number. This parameter can be a value from 1 to 15

### Return values

- **Host**: channel state This parameter can be one of these values: HC\_IDLE/ HC\_XFRC/ HC\_HALTED/ HC\_NYET/ HC\_NAK/ HC\_STALL/ HC\_XACTERR/ HC\_BBLERR/ HC\_DATATGLERR

### HAL\_HCD\_HC\_GetXferCount

### Function name

`uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * hhcd, uint8_t chnum)`

### Function description

Return the last host transfer size.

### Parameters

- **hhcd**: HCD handle
- **chnum**: Channel number. This parameter can be a value from 1 to 15

### Return values

- **last**: transfer size in byte

### HAL\_HCD\_GetCurrentFrame

### Function name

`uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)`

### Function description

Return the current Host frame number.

### Parameters

- **hhcd**: HCD handle

### Return values

- **Current**: Host frame number

### HAL\_HCD\_GetCurrentSpeed

### Function name

`uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)`

### Function description

Return the Host enumeration speed.

### Parameters

- **hhcd**: HCD handle

### Return values

- **Enumeration**: speed

## 35.3 HCD Firmware driver defines

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



**35.3.1 HCD**

HCD

***HCD Exported Macros***`__HAL_HCD_ENABLE``__HAL_HCD_DISABLE``__HAL_HCD_GET_FLAG``__HAL_HCD_CLEAR_FLAG``__HAL_HCD_IS_INVALID_INTERRUPT``__HAL_HCD_CLEAR_HC_INT``__HAL_HCD_MASK_HALT_HC_INT``__HAL_HCD_UNMASK_HALT_HC_INT``__HAL_HCD_MASK_ACK_HC_INT``__HAL_HCD_UNMASK_ACK_HC_INT`***HCD PHY Module***`HCD_PHY_ULPI``HCD_PHY_EMBEDDED`***HCD Speed***`HCD_SPEED_HIGH``HCD_SPEED_FULL``HCD_SPEED_LOW`

## 36 HAL I2C Generic Driver

### 36.1 I2C Firmware driver registers structures

#### 36.1.1 I2C\_InitTypeDef

*I2C\_InitTypeDef* is defined in the `stm32f4xx_hal_i2c.h`

##### Data Fields

- *uint32\_t* *ClockSpeed*
- *uint32\_t* *DutyCycle*
- *uint32\_t* *OwnAddress1*
- *uint32\_t* *AddressingMode*
- *uint32\_t* *DualAddressMode*
- *uint32\_t* *OwnAddress2*
- *uint32\_t* *GeneralCallMode*
- *uint32\_t* *NoStretchMode*

##### Field Documentation

- *uint32\_t I2C\_InitTypeDef::ClockSpeed*  
Specifies the clock frequency. This parameter must be set to a value lower than 400kHz
- *uint32\_t I2C\_InitTypeDef::DutyCycle*  
Specifies the I2C fast mode duty cycle. This parameter can be a value of [I2C\\_duty\\_cycle\\_in\\_fast\\_mode](#)
- *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::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](#)

#### 36.1.2 \_\_I2C\_HandleTypeDef

*\_\_I2C\_HandleTypeDef* is defined in the `stm32f4xx_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*
- *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 Devaddress**
- **\_\_IO uint32\_t Memaddress**
- **\_\_IO uint32\_t MemaddSize**
- **\_\_IO uint32\_t EventCount**

#### 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 transfer options
- **\_\_IO uint32\_t \_\_I2C\_HandleTypeDef::PreviousState**  
I2C communication Previous state and mode context for internal usage
- **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::Devaddress**  
I2C Target device address
- **\_\_IO uint32\_t \_\_I2C\_HandleTypeDef::Memaddress**  
I2C Target memory address
- **\_\_IO uint32\_t \_\_I2C\_HandleTypeDef::MemaddSize**  
I2C Target memory address size
- **\_\_IO uint32\_t \_\_I2C\_HandleTypeDef::EventCount**  
I2C Event counter

## 36.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

### 36.2.1 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 stream
    - Enable the DMAx interface clock using
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx stream
    - 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 stream
3. Configure the Communication Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, 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() 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()

### 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\_definition and are listed below:
  - I2C\_FIRST\_AND\_LAST\_FRAME: No sequential usage, functional 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 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 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 opposite 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()

#### **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()

#### **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\_GET\_FLAG: Checks 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

### 36.2.2 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:
  - Communication Speed
  - Duty cycle
  - Addressing mode
  - Own Address 1
  - Dual Addressing mode
  - Own Address 2
  - 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\(\)\*\*\*](#)

### 36.2.3 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()*

#### 36.2.4 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()*

#### 36.2.5 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
- **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 XferOptions definition

### 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 XferOptions definition

### 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 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 XferOptions definition

### 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 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 XferOptions definition

**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 XferOptions definition

### 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 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 XferOptions definition

### 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 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 XferOptions definition

### 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 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 XferOptions definition

### 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 XferDirection definition
- **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

## 36.3 I2C Firmware driver defines

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

### 36.3.1 I2C

I2C

*I2C addressing mode*

**I2C\_ADDRESSINGMODE\_7BIT**

**I2C\_ADDRESSINGMODE\_10BIT**

*I2C dual addressing mode*

I2C\_DUALADDRESS\_DISABLE

I2C\_DUALADDRESS\_ENABLE

*I2C duty cycle in fast mode*

I2C\_DUTYCYCLE\_2

I2C\_DUTYCYCLE\_16\_9

*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

AF 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\_WRONG\_START

Wrong start 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 or disable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - `I2C_IT_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

- None

### `__HAL_I2C_DISABLE_IT`

### `__HAL_I2C_GET_IT_SOURCE`

**Description:**

- Checks if the specified I2C interrupt source is enabled or disabled.

**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_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

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

## \_\_HAL\_I2C\_GET\_FLAG

### Description:

- Checks 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_OVR`: Overrun/Underrun flag
  - `I2C_FLAG_AF`: Acknowledge failure flag
  - `I2C_FLAG_ARLO`: Arbitration lost flag
  - `I2C_FLAG_BERR`: Bus error flag
  - `I2C_FLAG_TXE`: Data register empty flag
  - `I2C_FLAG_RXNE`: Data register not empty flag
  - `I2C_FLAG_STOPF`: Stop detection flag
  - `I2C_FLAG_ADD10`: 10-bit header sent flag
  - `I2C_FLAG_BTF`: Byte transfer finished flag
  - `I2C_FLAG_ADDR`: Address sent flag Address matched flag
  - `I2C_FLAG_SB`: Start bit flag
  - `I2C_FLAG_DUALF`: Dual flag
  - `I2C_FLAG_GENCALL`: General call header flag
  - `I2C_FLAG_TRA`: Transmitter/Receiver flag
  - `I2C_FLAG_BUSY`: Bus busy flag
  - `I2C_FLAG_MSL`: Master/Slave flag

### Return value:

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

## \_\_HAL\_I2C\_CLEAR\_FLAG

### Description:

- Clears the I2C pending flags which are cleared by writing 0 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_OVR`: Overrun/Underrun flag (Slave mode)
  - `I2C_FLAG_AF`: Acknowledge failure flag
  - `I2C_FLAG_ARLO`: Arbitration lost flag (Master mode)
  - `I2C_FLAG_BERR`: Bus error flag

### Return value:

- None

## \_\_HAL\_I2C\_CLEAR\_ADDRFLAG

### Description:

- Clears the I2C ADDR pending flag.

### Parameters:

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

### Return value:

- None

### \_\_HAL\_I2C\_CLEAR\_STOPFLAG

**Description:**

- Clears the I2C STOPF pending flag.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.

**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

***I2C Flag definition***

I2C\_FLAG\_OVR

I2C\_FLAG\_AF

I2C\_FLAG\_ARLO

I2C\_FLAG\_BERR

I2C\_FLAG\_TXE

I2C\_FLAG\_RXNE

I2C\_FLAG\_STOPF

I2C\_FLAG\_ADD10

I2C\_FLAG\_BTF

I2C\_FLAG\_ADDR

I2C\_FLAG\_SB

I2C\_FLAG\_DUALF

I2C\_FLAG\_GENCALL

I2C\_FLAG\_TRA

I2C\_FLAG\_BUSY

I2C\_FLAG\_MSL

*I2C general call addressing mode*

I2C\_GENERALCALL\_DISABLE

I2C\_GENERALCALL\_ENABLE

*I2C Interrupt configuration definition*

I2C\_IT\_BUF

I2C\_IT\_EVT

I2C\_IT\_ERR

*I2C Private macros to check input parameters*

IS\_I2C\_DUTY\_CYCLE

IS\_I2C\_ADDRESSING\_MODE

IS\_I2C\_DUAL\_ADDRESS

IS\_I2C\_GENERAL\_CALL

IS\_I2C\_NO\_STRETCH

IS\_I2C\_MEMADD\_SIZE

IS\_I2C\_CLOCK\_SPEED

IS\_I2C\_OWN\_ADDRESS1

IS\_I2C\_OWN\_ADDRESS2

IS\_I2C\_TRANSFER\_OPTIONS\_REQUEST

IS\_I2C\_TRANSFER\_OTHER\_OPTIONS\_REQUEST

I2C\_CHECK\_FLAG

I2C\_CHECK\_IT\_SOURCE

*I2C Memory Address Size*

I2C\_MEMADD\_SIZE\_8BIT

I2C\_MEMADD\_SIZE\_16BIT

*I2C nostretch mode*

I2C\_NOSTRETCH\_DISABLE

I2C\_NOSTRETCH\_ENABLE

*I2C XferDirection definition*

I2C\_DIRECTION\_RECEIVE

I2C\_DIRECTION\_TRANSMIT

*I2C XferOptions definition*

I2C\_FIRST\_FRAME

I2C\_FIRST\_AND\_NEXT\_FRAME

I2C\_NEXT\_FRAME

I2C\_FIRST\_AND\_LAST\_FRAME

I2C\_LAST\_FRAME\_NO\_STOP

I2C\_LAST\_FRAME

I2C\_OTHER\_FRAME

I2C\_OTHER\_AND\_LAST\_FRAME



## 37 HAL I2C Extension Driver

### 37.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

#### 37.1.1 I2C peripheral extension features

Comparing to other previous devices, the I2C interface for STM32F427xx/437xx/ 429xx/439xx devices contains the following additional features :

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter

#### 37.1.2 How to use this driver

This driver provides functions to configure Noise Filter

1. Configure I2C Analog noise filter using the function `HAL_I2C_AnalogFilter_Config()`
2. Configure I2C Digital noise filter using the function `HAL_I2C_DigitalFilter_Config()`

#### 37.1.3 Extension features functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- [`HAL\_I2CEx\_ConfigAnalogFilter\(\)`](#)
- [`HAL\_I2CEx\_ConfigDigitalFilter\(\)`](#)

#### 37.1.4 Detailed description of functions

##### HAL\_I2CEx\_ConfigAnalogFilter

###### Function name

`HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)`

###### Function description

Configures 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

Configures 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 0x00 and 0x0F.

### Return values

- **HAL**: status

## 37.2 I2CEX Firmware driver defines

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

### 37.2.1 I2CEX I2CEX *I2C Analog Filter*

I2C\_ANALOGFILTER\_ENABLE

I2C\_ANALOGFILTER\_DISABLE

## 38 HAL I2S Generic Driver

### 38.1 I2S Firmware driver registers structures

#### 38.1.1 I2S\_InitTypeDef

*I2S\_InitTypeDef* is defined in the `stm32f4xx_hal_i2s.h`

##### Data Fields

- *uint32\_t Mode*
- *uint32\_t Standard*
- *uint32\_t DataFormat*
- *uint32\_t MCLKOutput*
- *uint32\_t AudioFreq*
- *uint32\_t CPOL*
- *uint32\_t ClockSource*
- *uint32\_t FullDuplexMode*

##### 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](#)
- *uint32\_t I2S\_InitTypeDef::ClockSource*  
Specifies the I2S Clock Source. This parameter can be a value of [I2S\\_Clock\\_Source](#)
- *uint32\_t I2S\_InitTypeDef::FullDuplexMode*  
Specifies the I2S FullDuplex mode. This parameter can be a value of [I2S\\_FullDuplex\\_Mode](#)

#### 38.1.2 \_\_I2S\_HandleTypeDef

*\_\_I2S\_HandleTypeDef* is defined in the `stm32f4xx_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*
- *void(\* IrqHandlerISR*
- *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  $NbSamplesReceived = RxBufferSize - RxBufferCount$ )
- **`void(* __I2S_HandleTypeDef::IrqHandlerISR)(struct __I2S_HandleTypeDef *hi2s)`**  
I2S function pointer on IrqHandler
- **`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](#)

## 38.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

### 38.2.1 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:

- I2S PLL clock is configured or
  - External clock source is configured after setting correctly the define constant `EXTERNAL_CLOCK_VALUE` in the `stm32f4xx_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() In Slave mode, if HAL\_I2S\_DMAStop is used to stop the communication, an error HAL\_I2S\_ERROR\_BUSY\_LINE\_RX is raised as the master continue to transmit data. In this case \_\_HAL\_I2S\_FLUSH\_RX\_DR macro must be used to flush the remaining data inside DR register and avoid using Delnit/Init process for the next transfer.

### 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
- \_\_HAL\_I2S\_FLUSH\_RX\_DR: Read DR Register to Flush RX Data

*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
  - TxRxCpltCallback : I2S TxRx 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
  - TxRxCpltCallback : I2S TxRx 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.

### 38.2.2 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
  - Full duplex mode
- 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\(\)\*](#)

### 38.2.3 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\_DMABuffer()*
- *HAL\_I2S\_DMAResume()*
- *HAL\_I2S\_DMAStop()*
- *HAL\_I2S\_IRQHandler()*
- *HAL\_I2S\_TxHalfCpltCallback()*
- *HAL\_I2S\_TxCpltCallback()*
- *HAL\_I2S\_RxHalfCpltCallback()*
- *HAL\_I2S\_RxCpltCallback()*
- *HAL\_I2S\_ErrorCallback()*

### 38.2.4 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()*

### 38.2.5 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

### 38.3 I2S Firmware driver defines

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

#### 38.3.1 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 Clock Source Definition***

I2S\_CLOCK\_PLL

I2S\_CLOCK\_EXTERNAL

***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

## HAL\_I2S\_ERROR\_BUSY\_LINE\_RX

Busy Rx Line 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

### `__HAL_I2S_FLUSH_RX_DR`

**Description:**

- Flush the I2S DR Register.

**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 FullDuplex Mode***

`I2S_FULLDUPLEXMODE_DISABLE`

`I2S_FULLDUPLEXMODE_ENABLE`

***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

## 39 HAL I2S Extension Driver

### 39.1 I2SEx Firmware driver API description

The following section lists the various functions of the I2SEx library.

#### 39.1.1 I2S Extension features

1. In I2S full duplex mode, each SPI peripheral is able to manage sending and receiving data simultaneously using two data lines. Each SPI peripheral has an extended block called I2Sxext (i.e I2S2ext for SPI2 and I2S3ext for SPI3).
2. The extension block is not a full SPI IP, it is used only as I2S slave to implement full duplex mode. The extension block uses the same clock sources as its master.
3. Both I2Sx and I2Sx\_ext can be configured as transmitters or receivers.

*Note:* Only I2Sx can deliver SCK and WS to I2Sx\_ext in full duplex mode, where I2Sx can be I2S2 or I2S3.

#### 39.1.2 How to use this driver

Three operation modes are available within this driver :

##### Polling mode IO operation

- Send and receive in the same time an amount of data in blocking mode using HAL\_I2SEx\_TransmitReceive()

##### Interrupt mode IO operation

- Send and receive in the same time an amount of data in non blocking mode using HAL\_I2SEx\_TransmitReceive\_IT()
- At transmission/reception end of transfer HAL\_I2SEx\_TxRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2SEx\_TxRxCpltCallback
- 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 and receive an amount of data in non blocking mode (DMA) using HAL\_I2SEx\_TransmitReceive\_DMA()
- At transmission/reception end of transfer HAL\_I2SEx\_TxRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxRxCpltCallback
- 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
- \_\_HAL\_I2SEXT\_FLUSH\_RX\_DR: In Full-Duplex Slave mode, if HAL\_I2S\_DMASStop is used to stop the communication, an error HAL\_I2S\_ERROR\_BUSY\_LINE\_RX is raised as the master continue to transmit data. In this case \_\_HAL\_I2SEXT\_FLUSH\_RX\_DR macro must be used to flush the remaining data inside I2Sx and I2Sx\_ext DR registers and avoid using Delnit/Init process for the next transfer.

#### 39.1.3 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\_I2SEx\_TransmitReceive()

3. No-Blocking mode functions with Interrupt are :
  - HAL\_I2SEx\_TransmitReceive\_IT()
  - HAL\_I2SEx\_FullDuplex\_IRQHandler()
4. No-Blocking mode functions with DMA are :
  - HAL\_I2SEx\_TransmitReceive\_DMA()
5. A set of Transfer Complete Callback are provided in non Blocking mode:
  - HAL\_I2SEx\_TxRxCpltCallback()

This section contains the following APIs:

- [HAL\\_I2SEx\\_TransmitReceive\(\)](#)
- [HAL\\_I2SEx\\_TransmitReceive\\_IT\(\)](#)
- [HAL\\_I2SEx\\_TransmitReceive\\_DMA\(\)](#)
- [HAL\\_I2SEx\\_FullDuplex\\_IRQHandler\(\)](#)
- [HAL\\_I2SEx\\_TxRxHalfCpltCallback\(\)](#)
- [HAL\\_I2SEx\\_TxRxCpltCallback\(\)](#)

### 39.1.4 Detailed description of functions

#### HAL\_I2SEx\_TransmitReceive

##### Function name

**HAL\_StatusTypeDef HAL\_I2SEx\_TransmitReceive (I2S\_HandleTypeDef \* hi2s, uint16\_t \* pTxData, uint16\_t \* pRxData, uint16\_t Size, uint32\_t Timeout)**

##### Function description

Full-Duplex Transmit/Receive data in blocking mode.

##### Parameters

- **hi2s**: pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData**: a 16-bit pointer to the Transmit data buffer.
- **pRxData**: a 16-bit pointer to the Receive 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\_I2SEx\_TransmitReceive\_IT

##### Function name

**HAL\_StatusTypeDef HAL\_I2SEx\_TransmitReceive\_IT (I2S\_HandleTypeDef \* hi2s, uint16\_t \* pTxData, uint16\_t \* pRxData, uint16\_t Size)**

##### Function description

Full-Duplex Transmit/Receive data in non-blocking mode using Interrupt.

### Parameters

- **hi2s**: pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData**: a 16-bit pointer to the Transmit data buffer.
- **pRxData**: 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\_I2SEx\_TransmitReceive\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_I2SEx\_TransmitReceive\_DMA (I2S\_HandleTypeDef \* hi2s, uint16\_t \* pTxData, uint16\_t \* pRxData, uint16\_t Size)**

#### Function description

Full-Duplex Transmit/Receive data in non-blocking mode using DMA.

#### Parameters

- **hi2s**: pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData**: a 16-bit pointer to the Transmit data buffer.
- **pRxData**: 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\_I2SEx\_FullDuplex\_IRQHandler

#### Function name

**void HAL\_I2SEx\_FullDuplex\_IRQHandler (I2S\_HandleTypeDef \* hi2s)**

#### Function description

This function handles I2S/I2Sext interrupt requests in full-duplex mode.

#### Parameters

- **hi2s**: I2S handle

#### Return values

- **HAL**: status

### HAL\_I2SEx\_TxRxHalfCpltCallback

#### Function name

**void HAL\_I2SEx\_TxRxHalfCpltCallback (I2S\_HandleTypeDef \* hi2s)**

#### Function description

Tx and Rx Transfer half completed callback.

#### Parameters

- **hi2s:** I2S handle

#### Return values

- **None:**

### HAL\_I2SEx\_TxRxCpltCallback

#### Function name

**void HAL\_I2SEx\_TxRxCpltCallback (I2S\_HandleTypeDef \* hi2s)**

#### Function description

Tx and Rx Transfer completed callback.

#### Parameters

- **hi2s:** I2S handle

#### Return values

- **None:**

## 39.2 I2SEx Firmware driver defines

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

### 39.2.1 I2SEx

I2SEx

*I2S Extended Exported Macros*

#### I2SxEXT

##### \_\_HAL\_I2SEXT\_ENABLE

###### Description:

- Enable or disable the specified I2SExt peripheral.

###### Parameters:

- \_\_HANDLE\_\_: specifies the I2S Handle.

###### Return value:

- None

##### \_\_HAL\_I2SEXT\_DISABLE

### \_\_HAL\_I2SEXT\_ENABLE\_IT

**Description:**

- Enable or disable the specified I2SExt 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\_I2SEXT\_DISABLE\_IT

### \_\_HAL\_I2SEXT\_GET\_IT\_SOURCE

**Description:**

- Checks if the specified I2SExt 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\_I2SEXT\_GET\_FLAG

**Description:**

- Checks whether the specified I2SExt 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\_I2SEXT\_CLEAR\_OVRFLAG****Description:**

- Clears the I2SExt OVR pending flag.

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.

**Return value:**

- None

**\_\_HAL\_I2SEXT\_CLEAR\_UDRFLAG****Description:**

- Clears the I2SExt UDR pending flag.

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.

**Return value:**

- None

**\_\_HAL\_I2SEXT\_FLUSH\_RX\_DR****Description:**

- Flush the I2S and I2SExt DR Registers.

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.

**Return value:**

- None

## 40 HAL IRDA Generic Driver

### 40.1 IRDA Firmware driver registers structures

#### 40.1.1 IRDA\_InitTypeDef

*IRDA\_InitTypeDef* is defined in the `stm32f4xx_hal_irda.h`

##### Data Fields

- *uint32\_t* *BaudRate*
- *uint32\_t* *WordLength*
- *uint32\_t* *Parity*
- *uint32\_t* *Mode*
- *uint8\_t* *Prescaler*
- *uint32\_t* *IrDAMode*

##### Field Documentation

- *uint32\_t* *IRDA\_InitTypeDef::BaudRate*  
This member configures the IRDA communication baud rate. The baud rate is computed using the following formula:
  - $IntegerDivider = ((PCLKx) / (8 * (hirda->Init.BaudRate)))$
  - $FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 8) + 0.5$
- *uint32\_t* *IRDA\_InitTypeDef::WordLength*  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [IRDA\\_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\\_Mode](#)
- *uint8\_t* *IRDA\_InitTypeDef::Prescaler*  
Specifies the Prescaler value to be programmed in the IrDA low-power Baud Register, for defining pulse width on which burst acceptance/rejection will be decided. This value is used as divisor of system clock to achieve required pulse width.
- *uint32\_t* *IRDA\_InitTypeDef::IrDAMode*  
Specifies the IrDA mode This parameter can be a value of [IRDA\\_Low\\_Power](#)

#### 40.1.2 IRDA\_HandleTypeDef

*IRDA\_HandleTypeDef* is defined in the `stm32f4xx_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*
- *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
- **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

## 40.2 IRDA Firmware driver API description

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

### 40.2.1 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:
  - a. Enable the USARTx interface clock.
  - b. IRDA pins configuration:
    - Enable the clock for the IRDA GPIOs.
    - Configure IRDA pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_IRDA\_Transmit\_IT() and HAL\_IRDA\_Receive\_IT()) APIs:
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. 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 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 IRDA DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx stream.
    - Configure the IRDAx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Parity, IrDA Mode, Prescaler and Mode(Receiver/Transmitter) in the hirda 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 end of half 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 end of half 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
- Pause the DMA Transfer using HAL\_IRDA\_DMAPause()
- Resume the DMA Transfer using HAL\_IRDA\_DMAResume()
- Stop the DMA Transfer using HAL\_IRDA\_DMAStop()

#### **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 the specified IRDA interrupt has occurred or not

*Note:* You can refer to the IRDA HAL driver header file for more useful macros

#### **40.2.2 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.

*Note: If the parity is enabled, the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. The IRDA frame format depends on the frame length defined by the M bit (8-bits or 9-bits). For more details, refer to Table Frame formats in Section Universal synchronous asynchronous receiver transmitter (USART) of the corresponding reference manual.*

### 40.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous IrDA mode.

- For the asynchronous mode only these parameters can be configured:
  - BaudRate
  - WordLength
  - 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible IRDA frame formats.
  - Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
  - Mode: Receiver/transmitter modes
  - IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

The HAL\_IRDA\_Init() API follows IRDA 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\(\)](#)

### 40.2.4 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 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. (#) 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 (#) Blocking mode APIs are : (++) HAL\_IRDA\_Transmit() (++) HAL\_IRDA\_Receive() (#) Non Blocking mode APIs with Interrupt are : (++) HAL\_IRDA\_Transmit\_IT() (++) HAL\_IRDA\_Receive\_IT() (++) HAL\_IRDA\_IRQHandler() (#) Non Blocking mode functions with DMA are : (++) HAL\_IRDA\_Transmit\_DMA() (++) HAL\_IRDA\_Receive\_DMA() (++) HAL\_IRDA\_DMAPause() (++) HAL\_IRDA\_DMAResume() (++) HAL\_IRDA\_DMAStop() (#) 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() (#) 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() (#) 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() (#) 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\(\)](#)

#### 40.2.5 Peripheral State and Errors 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.
- [HAL\\_IRDA\\_GetError\(\)](#) check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL\\_IRDA\\_GetState\(\)](#)
- [HAL\\_IRDA\\_GetError\(\)](#)

#### 40.2.6 Detailed description of functions

##### HAL\_IRDA\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_IRDA\_Init (IRDA\_HandleTypeDef \* hirda)**

###### Function description

Initializes the IRDA mode according to the specified parameters in the IRDA\_InitTypeDef and create 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

Deinitializes 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\_Msplnit

###### Function name

**void HAL\_IRDA\_Msplnit (IRDA\_HandleTypeDef \* hirda)**

### Function description

IRDA MSP Init.

### 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

IRDA MSP DeInit.

### 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

Sends 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 non 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.

#### 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 non 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.

#### 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

Receives 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 data received contain the parity bit.

## HAL\_IRDA\_DMAPause

### Function name

**HAL\_StatusTypeDef HAL\_IRDA\_DMAPause (IRDA\_HandleTypeDef \* hirda)**

### Function description

Pauses 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

Resumes 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\_DMAStop

### Function name

`HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)`

### Function description

Stops 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\_Abort

### Function name

`HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)`

### Function description

Abort ongoing transfers (blocking mode).

### Parameters

- **hirda**: IRDA 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 PPP InterruptsDisable 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**: IRDA 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 PPP Interrupts  
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**: IRDA 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 PPP Interrupts  
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**: IRDA 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 PPP Interrupts  
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 READY  
At 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**: IRDA 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 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 READY  
At 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**: IRDA 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 PPP Interrupts  
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 READY  
At 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**

This function handles 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 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\_RxCpltCallback**
**Function name**

```
void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)
```

**Function description**

Rx 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\_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 Transmit 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 state.

### Parameters

- **hirda**: Pointer to a `IRDA_HandleTypeDef` structure that contains the configuration information for the specified IRDA.

### Return values

- **HAL**: state

**HAL\_IRDA\_GetError**

### Function name

`uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)`

### Function description

Return the IRDA error code.

### Parameters

- **hirda**: Pointer to a `IRDA_HandleTypeDef` structure that contains the configuration information for the specified IRDA.

### Return values

- **IRDA**: Error Code

## 40.3 IRDA Firmware driver defines

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

### 40.3.1 IRDA

IRDA

**IRDA Error Code**

#### 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

**IRDA Exported Macros**
**\_\_HAL\_IRDA\_RESET\_HANDLE\_STATE**
**Description:**

- Reset IRDA handle gstate & RxState.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**\_\_HAL\_IRDA\_FLUSH\_DRREGISTER**
**Description:**

- Flush the IRDA DR register.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**\_\_HAL\_IRDA\_GET\_FLAG**
**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `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_IDLE`: Idle Line detection 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\_CLEAR\_FLAG**

**Description:**

- Clear the specified IRDA pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA\_FLAG\_TC: Transmission Complete flag.
  - IRDA\_FLAG\_RXNE: Receive data register not empty flag.

**Return value:**

- None

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

### **\_\_HAL\_IRDA\_CLEAR\_PFLAG**

**Description:**

- Clear the IRDA PE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### **\_\_HAL\_IRDA\_CLEAR\_FEFLAG**

**Description:**

- Clear the IRDA FE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### **\_\_HAL\_IRDA\_CLEAR\_NEFLAG**

**Description:**

- Clear the IRDA NE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**\_\_HAL\_IRDA\_CLEAR\_OREFLAG**
**Description:**

- Clear the IRDA ORE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**\_\_HAL\_IRDA\_CLEAR\_IDLEFLAG**
**Description:**

- Clear the IRDA IDLE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**\_\_HAL\_IRDA\_ENABLE\_IT**
**Description:**

- Enable the specified IRDA interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_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. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_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\_SOURCE**

**Description:**

- Check whether the specified IRDA interrupt has occurred or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_IT\_\_**: 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: Error interrupt
  - IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

- The: new state of **\_\_IT\_\_** (TRUE or FALSE).

### **\_\_HAL\_IRDA\_ONE\_BIT\_SAMPLE\_ENABLE**

**Description:**

- Macro to enable the IRDA's one bit sample method.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle.

**Return value:**

- None

### **\_\_HAL\_IRDA\_ONE\_BIT\_SAMPLE\_DISABLE**

**Description:**

- Macro to disable the IRDA's 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. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**\_\_HAL\_IRDA\_DISABLE**
**Description:**

- Disable UART/USART associated to IRDA Handle.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

**IRDA Flags**

IRDA\_FLAG\_TXE

IRDA\_FLAG\_TC

IRDA\_FLAG\_RXNE

IRDA\_FLAG\_IDLE

IRDA\_FLAG\_ORE

IRDA\_FLAG\_NE

IRDA\_FLAG\_FE

IRDA\_FLAG\_PE

**IRDA Interrupt Definitions**

IRDA\_IT\_PE

IRDA\_IT\_TXE

IRDA\_IT\_TC

IRDA\_IT\_RXNE

IRDA\_IT\_IDLE

IRDA\_IT\_LBD

IRDA\_IT\_CTS

IRDA\_IT\_ERR

**IRDA Low Power**

IRDA\_POWERMODE\_LOWPOWER

IRDA\_POWERMODE\_NORMAL

*IRDA Transfer Mode*

IRDA\_MODE\_RX

IRDA\_MODE\_TX

IRDA\_MODE\_TX\_RX

*IRDA Parity*

IRDA\_PARITY\_NONE

IRDA\_PARITY\_EVEN

IRDA\_PARITY\_ODD

*IRDA Word Length*

IRDA\_WORDLENGTH\_8B

IRDA\_WORDLENGTH\_9B



## 41 HAL IWDG Generic Driver

### 41.1 IWDG Firmware driver registers structures

#### 41.1.1 IWDG\_InitTypeDef

*IWDG\_InitTypeDef* is defined in the stm32f4xx\_hal\_iwdg.h

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*

##### 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

#### 41.1.2 IWDG\_HandleTypeDef

*IWDG\_HandleTypeDef* is defined in the stm32f4xx\_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

### 41.2 IWDG Firmware driver API description

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

#### 41.2.1 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 (0xFFFF). 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. STM32F4xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM5 CH4 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

### 41.2.2 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 and IWDG\_RLR.
  - 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.
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

### 41.2.3 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.
- 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\(\)\*](#)

### 41.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- [\*HAL\\_IWDG\\_Refresh\(\)\*](#)

### 41.2.5 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

## 41.3 IWDG Firmware driver defines

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

### 41.3.1 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 and IWDG\_RLR 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

## 42 HAL LPTIM Generic Driver

### 42.1 LPTIM Firmware driver registers structures

#### 42.1.1 LPTIM\_ClockConfigTypeDef

*LPTIM\_ClockConfigTypeDef* is defined in the `stm32f4xx_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](#)

#### 42.1.2 LPTIM\_ULPClockConfigTypeDef

*LPTIM\_ULPClockConfigTypeDef* is defined in the `stm32f4xx_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](#)

#### 42.1.3 LPTIM\_TriggerConfigTypeDef

*LPTIM\_TriggerConfigTypeDef* is defined in the `stm32f4xx_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](#)

#### 42.1.4 LPTIM\_InitTypeDef

*LPTIM\_InitTypeDef* is defined in the `stm32f4xx_hal_lptim.h`

**Data Fields**

- ***LPTIM\_ClockConfigTypeDef*** *Clock*
- ***LPTIM\_ULPClockConfigTypeDef*** *UltraLowPowerClock*
- ***LPTIM\_TriggerConfigTypeDef*** *Trigger*
- ***uint32\_t*** *OutputPolarity*
- ***uint32\_t*** *UpdateMode*
- ***uint32\_t*** *CounterSource*

#### 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*

### 42.1.5

#### LPTIM\_HandleTypeDef

*LPTIM\_HandleTypeDef* is defined in the *stm32f4xx\_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

## 42.2

### LPTIM Firmware driver API description

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

#### 42.2.1

##### 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.
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.

### 42.2.2 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\(\)\*](#)

### 42.2.3 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()*

#### 42.2.4 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()*

#### 42.2.5 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()*

#### 42.2.6 Detailed description of functions

##### HAL\_LPTIM\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_LPTIM\_Init (LPTIM\_HandleTypeDef \* hltim)**

###### Function description

Initialize the LPTIM according to the specified parameters in the LPTIM\_InitTypeDef and initialize the associated handle.

###### Parameters

- **hltim**: 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 \* hltim)**
**Function description**

Stop the LPTIM PWM generation.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **HAL:** status

**HAL\_LPTIM\_PWM\_Start\_IT**
**Function name**
**HAL\_StatusTypeDef HAL\_LPTIM\_PWM\_Start\_IT (LPTIM\_HandleTypeDef \* hltim, uint32\_t Period, uint32\_t Pulse)**
**Function description**

Start the LPTIM PWM generation in interrupt mode.

**Parameters**

- **hltim:** 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 \* hltim)**
**Function description**

Stop the LPTIM PWM generation in interrupt mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **HAL:** status

**HAL\_LPTIM\_OnePulse\_Start**
**Function name**
**HAL\_StatusTypeDef HAL\_LPTIM\_OnePulse\_Start (LPTIM\_HandleTypeDef \* hltim, 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 \* hltim)**

**Function description**

Handle LPTIM interrupt request.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_CompareMatchCallback

**Function name**

**void HAL\_LPTIM\_CompareMatchCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Compare match callback in non-blocking mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_AutoReloadMatchCallback

**Function name**

**void HAL\_LPTIM\_AutoReloadMatchCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Autoreload match callback in non-blocking mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_TriggerCallback

**Function name**

**void HAL\_LPTIM\_TriggerCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Trigger detected callback in non-blocking mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_CompareWriteCallback

**Function name**

**void HAL\_LPTIM\_CompareWriteCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Compare write callback in non-blocking mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_AutoReloadWriteCallback

**Function name**

**void HAL\_LPTIM\_AutoReloadWriteCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Autoreload write callback in non-blocking mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_DirectionUpCallback

**Function name**

**void HAL\_LPTIM\_DirectionUpCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Direction counter changed from Down to Up callback in non-blocking mode.

**Parameters**

- **hltim:** LPTIM handle

**Return values**

- **None:**

### HAL\_LPTIM\_DirectionDownCallback

**Function name**

**void HAL\_LPTIM\_DirectionDownCallback (LPTIM\_HandleTypeDef \* hltim)**

**Function description**

Direction counter changed from Up to Down callback in non-blocking mode.

**Parameters**

- **hltim:** 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.

## 42.3 LPTIM Firmware driver defines

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

### 42.3.1 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\_APBCLK\_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\_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\_OPTR\_CONFIG

### Description:

- LPTIM Option Register.

### Parameters:

- `__HANDLE__`: LPTIM handle
- `__VALUE__`: This parameter can be a value of :
  - `LPTIM_OP_PAD_AF`
  - `LPTIM_OP_PAD_PA4`
  - `LPTIM_OP_PAD_PB9`
  - `LPTIM_OP_TIM_DAC`

### Return value:

- None



#### \_\_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.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_ENABLE\_FALLING\_EDGE

**Description:**

- Enable falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_DISABLE\_FALLING\_EDGE

**Description:**

- Disable falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_ENABLE\_RISING\_EDGE

**Description:**

- Enable rising edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_DISABLE\_RISING\_EDGE

**Description:**

- Disable rising edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE

**Description:**

- Enable rising & falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_DISABLE\_RISING\_FALLING\_EDGE

**Description:**

- Disable rising & falling edge trigger on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_GET\_FLAG

**Description:**

- Check whether the LPTIM Wake-up Timer associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_CLEAR\_FLAG

**Description:**

- Clear the LPTIM Wake-up Timer associated Exti line flag.

**Return value:**

- None.

#### \_\_HAL\_LPTIM\_WAKEUPTIMER\_EXTI\_GENERATE\_SWIT

**Description:**

- Generate a Software interrupt on the LPTIM Wake-up Timer associated Exti line.

**Return value:**

- None.

**LPTIM Exported Types**

#### LPTIM\_EXTI\_LINE\_WAKEUPTIMER\_EVENT

External interrupt line 23 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 Interrupts Definition***

LPTIM\_IT\_DOWN

LPTIM\_IT\_UP

LPTIM\_IT\_ARROK

LPTIM\_IT\_CMPOK

LPTIM\_IT\_EXTTRIG

LPTIM\_IT\_ARRM

LPTIM\_IT\_CMPM

***Register Definition***

LPTIM\_OP\_PAD\_AF

LPTIM\_OP\_PAD\_PA4

LPTIM\_OP\_PAD\_PB9

LPTIM\_OP\_TIM\_DAC

***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 Updating Mode***

LPTIM\_UPDATE\_IMMEDIATE

LPTIM\_UPDATE\_ENDOFPERIOD

## 43 HAL LTDC Generic Driver

### 43.1 LTDC Firmware driver registers structures

#### 43.1.1 LTDC\_ColorTypeDef

*LTDC\_ColorTypeDef* is defined in the `stm32f4xx_hal_ltdc.h`

##### Data Fields

- *uint8\_t Blue*
- *uint8\_t Green*
- *uint8\_t Red*
- *uint8\_t Reserved*

##### Field Documentation

- *uint8\_t LTDC\_ColorTypeDef::Blue*  
Configures the blue value. This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF`.
- *uint8\_t LTDC\_ColorTypeDef::Green*  
Configures the green value. This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF`.
- *uint8\_t LTDC\_ColorTypeDef::Red*  
Configures the red value. This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF`.
- *uint8\_t LTDC\_ColorTypeDef::Reserved*  
Reserved `0xFF`

#### 43.1.2 LTDC\_InitTypeDef

*LTDC\_InitTypeDef* is defined in the `stm32f4xx_hal_ltdc.h`

##### Data Fields

- *uint32\_t HSPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t DEPolarity*
- *uint32\_t PCPolarity*
- *uint32\_t HorizontalSync*
- *uint32\_t VerticalSync*
- *uint32\_t AccumulatedHBP*
- *uint32\_t AccumulatedVBP*
- *uint32\_t AccumulatedActiveW*
- *uint32\_t AccumulatedActiveH*
- *uint32\_t TotalWidth*
- *uint32\_t TotalHeigh*
- *LTDC\_ColorTypeDef Backcolor*

##### Field Documentation

- *uint32\_t LTDC\_InitTypeDef::HSPolarity*  
configures the horizontal synchronization polarity. This parameter can be one value of [LTDC\\_HS\\_POLARITY](#)
- *uint32\_t LTDC\_InitTypeDef::VSPolarity*  
configures the vertical synchronization polarity. This parameter can be one value of [LTDC\\_VS\\_POLARITY](#)
- *uint32\_t LTDC\_InitTypeDef::DEPolarity*  
configures the data enable polarity. This parameter can be one of value of [LTDC\\_DE\\_POLARITY](#)

- ***uint32\_t LTDC\_InitTypeDef::PCPolarity***  
configures the pixel clock polarity. This parameter can be one of value of ***LTDC\_PC\_POLARITY***
- ***uint32\_t LTDC\_InitTypeDef::HorizontalSync***  
configures the number of Horizontal synchronization width. This parameter must be a number between ***Min\_Data = 0x000*** and ***Max\_Data = 0xFFF***.
- ***uint32\_t LTDC\_InitTypeDef::VerticalSync***  
configures the number of Vertical synchronization height. This parameter must be a number between ***Min\_Data = 0x000*** and ***Max\_Data = 0x7FF***.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedHBP***  
configures the accumulated horizontal back porch width. This parameter must be a number between ***Min\_Data = LTDC\_HorizontalSync*** and ***Max\_Data = 0xFFF***.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedVBP***  
configures the accumulated vertical back porch height. This parameter must be a number between ***Min\_Data = LTDC\_VerticalSync*** and ***Max\_Data = 0x7FF***.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedActiveW***  
configures the accumulated active width. This parameter must be a number between ***Min\_Data = LTDC\_AccumulatedHBP*** and ***Max\_Data = 0xFFF***.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedActiveH***  
configures the accumulated active height. This parameter must be a number between ***Min\_Data = LTDC\_AccumulatedVBP*** and ***Max\_Data = 0x7FF***.
- ***uint32\_t LTDC\_InitTypeDef::TotalWidth***  
configures the total width. This parameter must be a number between ***Min\_Data = LTDC\_AccumulatedActiveW*** and ***Max\_Data = 0xFFF***.
- ***uint32\_t LTDC\_InitTypeDef::TotalHeigh***  
configures the total height. This parameter must be a number between ***Min\_Data = LTDC\_AccumulatedActiveH*** and ***Max\_Data = 0x7FF***.
- ***LTDC\_ColorTypeDef LTDC\_InitTypeDef::BackColor***  
Configures the background color.

### 43.1.3

#### LTDC\_LayerCfgTypeDef

***LTDC\_LayerCfgTypeDef*** is defined in the `stm32f4xx_hal_ltdc.h`

##### Data Fields

- ***uint32\_t WindowX0***
- ***uint32\_t WindowX1***
- ***uint32\_t WindowY0***
- ***uint32\_t WindowY1***
- ***uint32\_t PixelFormat***
- ***uint32\_t Alpha***
- ***uint32\_t Alpha0***
- ***uint32\_t BlendingFactor1***
- ***uint32\_t BlendingFactor2***
- ***uint32\_t FBStartAdress***
- ***uint32\_t ImageWidth***
- ***uint32\_t ImageHeight***
- ***LTDC\_ColorTypeDef Backcolor***

##### Field Documentation

- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowX0***  
Configures the Window Horizontal Start Position. This parameter must be a number between ***Min\_Data = 0x000*** and ***Max\_Data = 0xFFF***.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowX1***  
Configures the Window Horizontal Stop Position. This parameter must be a number between ***Min\_Data = 0x000*** and ***Max\_Data = 0xFFF***.

- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowY0***  
Configures the Window vertical Start Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0x7FF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowY1***  
Configures the Window vertical Stop Position. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x7FF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::PixelFormat***  
Specifies the pixel format. This parameter can be one of value of ***LTDC\_Pixelformat***
- ***uint32\_t LTDC\_LayerCfgTypeDef::Alpha***  
Specifies the constant alpha used for blending. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::Alpha0***  
Configures the default alpha value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::BlendingFactor1***  
Select the blending factor 1. This parameter can be one of value of ***LTDC\_BlendingFactor1***
- ***uint32\_t LTDC\_LayerCfgTypeDef::BlendingFactor2***  
Select the blending factor 2. This parameter can be one of value of ***LTDC\_BlendingFactor2***
- ***uint32\_t LTDC\_LayerCfgTypeDef::FBStartAddress***  
Configures the color frame buffer address
- ***uint32\_t LTDC\_LayerCfgTypeDef::ImageWidth***  
Configures the color frame buffer line length. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x1FFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::ImageHeight***  
Specifies the number of line in frame buffer. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0x7FF.
- ***LTDC\_ColorTypeDef LTDC\_LayerCfgTypeDef::BackColor***  
Configures the layer background color.

#### 43.1.4

#### **LTDC\_HandleTypeDef**

**LTDC\_HandleTypeDef** is defined in the stm32f4xx\_hal\_ltdc.h

##### Data Fields

- ***LTDC\_TypeDef \* Instance***
- ***LTDC\_InitTypeDef Init***
- ***LTDC\_LayerCfgTypeDef LayerCfg***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_LTDC\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

##### Field Documentation

- ***LTDC\_TypeDef\* LTDC\_HandleTypeDef::Instance***  
LTDC Register base address
- ***LTDC\_InitTypeDef LTDC\_HandleTypeDef::Init***  
LTDC parameters
- ***LTDC\_LayerCfgTypeDef LTDC\_HandleTypeDef::LayerCfg[MAX\_LAYER]***  
LTDC Layers parameters
- ***HAL\_LockTypeDef LTDC\_HandleTypeDef::Lock***  
LTDC Lock
- ***\_\_IO HAL\_LTDC\_StateTypeDef LTDC\_HandleTypeDef::State***  
LTDC state
- ***\_\_IO uint32\_t LTDC\_HandleTypeDef::ErrorCode***  
LTDC Error code

## 43.2 LTDC Firmware driver API description

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

### 43.2.1 How to use this driver

The LTDC HAL driver can be used as follows:

1. Declare a LTDC\_HandleTypeDef handle structure, for example: LTDC\_HandleTypeDef hltdc;
2. Initialize the LTDC low level resources by implementing the HAL\_LTDC\_MspInit() API:
  - a. Enable the LTDC interface clock
  - b. NVIC configuration if you need to use interrupt process
    - Configure the LTDC interrupt priority
    - Enable the NVIC LTDC IRQ Channel
3. Initialize the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using HAL\_LTDC\_Init() function

#### Configuration

1. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using HAL\_LTDC\_ConfigLayer() function for foreground or/and background layer.
2. Optionally, configure and enable the CLUT using HAL\_LTDC\_ConfigCLUT() and HAL\_LTDC\_EnableCLUT functions.
3. Optionally, enable the Dither using HAL\_LTDC\_EnableDither().
4. Optionally, configure and enable the Color keying using HAL\_LTDC\_ConfigColorKeying() and HAL\_LTDC\_EnableColorKeying functions.
5. Optionally, configure LineInterrupt using HAL\_LTDC\_ProgramLineEvent() function
6. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions: HAL\_LTDC\_SetPixelFormat(), HAL\_LTDC\_SetAlpha(), HAL\_LTDC\_SetWindowSize(), HAL\_LTDC\_SetWindowPosition() and HAL\_LTDC\_SetAddress().
7. Variant functions with \_NoReload suffix allows to set the LTDC configuration/settings without immediate reload. This is useful in case when the program requires to modify several LTDC settings (on one or both layers) then applying(reload) these settings in one shot by calling the function HAL\_LTDC\_Reload(). After calling the \_NoReload functions to set different color/format/layer settings, the program shall call the function HAL\_LTDC\_Reload() to apply(reload) these settings. Function HAL\_LTDC\_Reload() can be called with the parameter ReloadType set to LTDC\_RELOAD\_IMMEDIATE if an immediate reload is required. Function HAL\_LTDC\_Reload() can be called with the parameter ReloadType set to LTDC\_RELOAD\_VERTICAL\_BLANKING if the reload should be done in the next vertical blanking period, this option allows to avoid display flicker by applying the new settings during the vertical blanking period.
8. To control LTDC state you can use the following function: HAL\_LTDC\_GetState()

#### LTDC HAL driver macros list

Below the list of most used macros in LTDC HAL driver.

- `__HAL_LTDC_ENABLE`: Enable the LTDC.
- `__HAL_LTDC_DISABLE`: Disable the LTDC.
- `__HAL_LTDC_LAYER_ENABLE`: Enable an LTDC Layer.
- `__HAL_LTDC_LAYER_DISABLE`: Disable an LTDC Layer.
- `__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG`: Reload Layer Configuration.
- `__HAL_LTDC_GET_FLAG`: Get the LTDC pending flags.
- `__HAL_LTDC_CLEAR_FLAG`: Clear the LTDC pending flags.
- `__HAL_LTDC_ENABLE_IT`: Enable the specified LTDC interrupts.
- `__HAL_LTDC_DISABLE_IT`: Disable the specified LTDC interrupts.
- `__HAL_LTDC_GET_IT_SOURCE`: Check whether the specified LTDC interrupt has occurred or not.

*Note:* You can refer to the LTDC HAL driver header file for more useful macros



### Callback registration

The compilation define `USE_HAL_LTDC_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use function `HAL_LTDC_RegisterCallback()` to register a callback.

Function `HAL_LTDC_RegisterCallback()` allows to register following callbacks:

- `LineEventCallback` : LTDC Line Event Callback.
- `ReloadEventCallback` : LTDC Reload Event Callback.
- `ErrorCallback` : LTDC Error Callback
- `MspInitCallback` : LTDC MspInit.
- `MspDeInitCallback` : LTDC MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function `HAL_LTDC_UnRegisterCallback()` to reset a callback to the default weak function.

`HAL_LTDC_UnRegisterCallback()` takes as parameters the HAL peripheral handle and the callback ID.

This function allows to reset following callbacks:

- `LineEventCallback` : LTDC Line Event Callback
- `ReloadEventCallback` : LTDC Reload Event Callback
- `ErrorCallback` : LTDC Error Callback
- `MspInitCallback` : LTDC MspInit
- `MspDeInitCallback` : LTDC MspDeInit.

By default, after the `HAL_LTDC_Init` and when the state is `HAL_LTDC_STATE_RESET` all callbacks are set to the corresponding weak functions: examples `HAL_LTDC_LineEventCallback()`, `HAL_LTDC_ErrorCallback()`. Exception done for `MspInit` and `MspDeInit` functions that are reset to the legacy weak (surcharged) functions in the `HAL_LTDC_Init()` and `HAL_LTDC_DeInit()` only when these callbacks are null (not registered beforehand). If not, `MspInit` or `MspDeInit` are not null, the `HAL_LTDC_Init()` and `HAL_LTDC_DeInit()` keep and use the user `MspInit/MspDeInit` callbacks (registered beforehand).

Callbacks can be registered/unregistered in `HAL_LTDC_STATE_READY` state only. Exception done `MspInit/MspDeInit` that can be registered/unregistered in `HAL_LTDC_STATE_READY` or `HAL_LTDC_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 `HAL_LTDC_RegisterCallback()` before calling `HAL_LTDC_DeInit()` or `HAL_LTDC_Init()` function.

When the compilation define `USE_HAL_LTDC_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.

### 43.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC
- De-initialize the LTDC

This section contains the following APIs:

- [\*HAL\\_LTDC\\_Init\(\)\*](#)
- [\*HAL\\_LTDC\\_DeInit\(\)\*](#)
- [\*HAL\\_LTDC\\_MspInit\(\)\*](#)
- [\*HAL\\_LTDC\\_MspDeInit\(\)\*](#)
- [\*HAL\\_LTDC\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_LTDC\\_LineEventCallback\(\)\*](#)
- [\*HAL\\_LTDC\\_ReloadEventCallback\(\)\*](#)

### 43.2.3 IO operation functions

This section provides function allowing to:

- Handle LTDC interrupt request

This section contains the following APIs:

- *HAL\_LTDC\_IRQHandler()*
- *HAL\_LTDC\_ErrorCallback()*
- *HAL\_LTDC\_LineEventCallback()*
- *HAL\_LTDC\_ReloadEventCallback()*

#### 43.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.

This section contains the following APIs:

- *HAL\_LTDC\_ConfigLayer()*
- *HAL\_LTDC\_ConfigColorKeying()*
- *HAL\_LTDC\_ConfigCLUT()*
- *HAL\_LTDC\_EnableColorKeying()*
- *HAL\_LTDC\_DisableColorKeying()*
- *HAL\_LTDC\_EnableCLUT()*
- *HAL\_LTDC\_DisableCLUT()*
- *HAL\_LTDC\_EnableDither()*
- *HAL\_LTDC\_DisableDither()*
- *HAL\_LTDC\_SetWindowSize()*
- *HAL\_LTDC\_SetWindowPosition()*
- *HAL\_LTDC\_SetPixelFormat()*
- *HAL\_LTDC\_SetAlpha()*
- *HAL\_LTDC\_SetAddress()*
- *HAL\_LTDC\_SetPitch()*
- *HAL\_LTDC\_ProgramLineEvent()*
- *HAL\_LTDC\_Reload()*
- *HAL\_LTDC\_ConfigLayer\_NoReload()*
- *HAL\_LTDC\_SetWindowSize\_NoReload()*
- *HAL\_LTDC\_SetWindowPosition\_NoReload()*
- *HAL\_LTDC\_SetPixelFormat\_NoReload()*
- *HAL\_LTDC\_SetAlpha\_NoReload()*
- *HAL\_LTDC\_SetAddress\_NoReload()*
- *HAL\_LTDC\_SetPitch\_NoReload()*
- *HAL\_LTDC\_ConfigColorKeying\_NoReload()*
- *HAL\_LTDC\_EnableColorKeying\_NoReload()*
- *HAL\_LTDC\_DisableColorKeying\_NoReload()*
- *HAL\_LTDC\_EnableCLUT\_NoReload()*
- *HAL\_LTDC\_DisableCLUT\_NoReload()*

#### 43.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC handle state.
- Get the LTDC handle error code.

This section contains the following APIs:

- [HAL\\_LTDC\\_GetState\(\)](#)
- [HAL\\_LTDC\\_GetError\(\)](#)

### 43.2.6 Detailed description of functions

#### HAL\_LTDC\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_Init (LTDC\_HandleTypeDef \* hltdc)**

##### Function description

Initialize the LTDC according to the specified parameters in the LTDC\_InitTypeDef.

##### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

##### Return values

- **HAL**: status

#### HAL\_LTDC\_DeInit

##### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_DeInit (LTDC\_HandleTypeDef \* hltdc)**

##### Function description

De-initialize the LTDC peripheral.

##### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

##### Return values

- **None**:

#### HAL\_LTDC\_MspInit

##### Function name

**void HAL\_LTDC\_MspInit (LTDC\_HandleTypeDef \* hltdc)**

##### Function description

Initialize the LTDC MSP.

##### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

##### Return values

- **None**:

#### HAL\_LTDC\_MspDeInit

##### Function name

**void HAL\_LTDC\_MspDeInit (LTDC\_HandleTypeDef \* hltdc)**

### Function description

De-initialize the LTDC MSP.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

### Return values

- **None**:

**HAL\_LTDC\_ErrorCallback**

### Function name

**void HAL\_LTDC\_ErrorCallback (LTDC\_HandleTypeDef \* hltdc)**

### Function description

Error LTDC callback.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

### Return values

- **None**:

**HAL\_LTDC\_LineEventCallback**

### Function name

**void HAL\_LTDC\_LineEventCallback (LTDC\_HandleTypeDef \* hltdc)**

### Function description

Line Event callback.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

### Return values

- **None**:

**HAL\_LTDC\_ReloadEventCallback**

### Function name

**void HAL\_LTDC\_ReloadEventCallback (LTDC\_HandleTypeDef \* hltdc)**

### Function description

Reload Event callback.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

### Return values

- **None**:

**HAL\_LTDC\_IRQHandler**

### Function name

**void HAL\_LTDC\_IRQHandler (LTDC\_HandleTypeDef \* hltdc)**

### Function description

Handle LTDC interrupt request.

### Parameters

- **hltdc**: pointer to a `LTDC_HandleTypeDef` structure that contains the configuration information for the LTDC.

### Return values

- **HAL**: status

### HAL\_LTDC\_ConfigLayer

### Function name

`HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)`

### Function description

Configure the LTDC Layer according to the specified parameters in the `LTDC_InitTypeDef` and create the associated handle.

### Parameters

- **hltdc**: pointer to a `LTDC_HandleTypeDef` structure that contains the configuration information for the LTDC.
- **pLayerCfg**: pointer to a `LTDC_LayerCfgTypeDef` structure that contains the configuration information for the Layer.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: `LTDC_LAYER_1` (0) or `LTDC_LAYER_2` (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_SetWindowSize

### Function name

`HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)`

### Function description

Set the LTDC window size.

### Parameters

- **hltdc**: pointer to a `LTDC_HandleTypeDef` structure that contains the configuration information for the LTDC.
- **XSize**: LTDC Pixel per line
- **YSize**: LTDC Line number
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: `LTDC_LAYER_1` (0) or `LTDC_LAYER_2` (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_SetWindowPosition

### Function name

`HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)`

### Function description

Set the LTDC window position.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0**: LTDC window X offset
- **Y0**: LTDC window Y offset
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_SetPixelFormat

#### Function name

HAL\_StatusTypeDef HAL\_LTDC\_SetPixelFormat (LTDC\_HandleTypeDef \* hltdc, uint32\_t Pixelformat, uint32\_t LayerIdx)

#### Function description

Reconfigure the pixel format.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Pixelformat**: new pixel format value.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1).

### Return values

- **HAL**: status

### HAL\_LTDC\_SetAlpha

#### Function name

HAL\_StatusTypeDef HAL\_LTDC\_SetAlpha (LTDC\_HandleTypeDef \* hltdc, uint32\_t Alpha, uint32\_t LayerIdx)

#### Function description

Reconfigure the layer alpha value.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha**: new alpha value.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_SetAddress

#### Function name

HAL\_StatusTypeDef HAL\_LTDC\_SetAddress (LTDC\_HandleTypeDef \* hltdc, uint32\_t Address, uint32\_t LayerIdx)

#### Function description

Reconfigure the frame buffer Address.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address**: new address value.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1).

### Return values

- **HAL**: status

### HAL\_LTDC\_SetPitch

### Function name

HAL\_StatusTypeDef HAL\_LTDC\_SetPitch (LTDC\_HandleTypeDef \* hltdc, uint32\_t LinePitchInPixels, uint32\_t LayerIdx)

### Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels**: New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx**: LTDC layer index concerned by the modification of line pitch.

### Return values

- **HAL**: status

### Notes

- This function should be called only after a previous call to HAL\_LTDC\_ConfigLayer() to modify the default pitch configured by HAL\_LTDC\_ConfigLayer() when required (refer to example described just above).

### HAL\_LTDC\_ConfigColorKeying

### Function name

HAL\_StatusTypeDef HAL\_LTDC\_ConfigColorKeying (LTDC\_HandleTypeDef \* hltdc, uint32\_t RGBValue, uint32\_t LayerIdx)

### Function description

Configure the color keying.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue**: the color key value
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_ConfigCLUT

### Function name

HAL\_StatusTypeDef HAL\_LTDC\_ConfigCLUT (LTDC\_HandleTypeDef \* hltdc, uint32\_t \* pCLUT, uint32\_t CLUTSize, uint32\_t LayerIdx)

### Function description

Load the color lookup table.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pCLUT**: pointer to the color lookup table address.
- **CLUTSize**: the color lookup table size.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL**: status

**HAL\_LTDC\_EnableColorKeying**
**Function name**

HAL\_StatusTypeDef HAL\_LTDC\_EnableColorKeying (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)

**Function description**

Enable the color keying.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL**: status

**HAL\_LTDC\_DisableColorKeying**
**Function name**

HAL\_StatusTypeDef HAL\_LTDC\_DisableColorKeying (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)

**Function description**

Disable the color keying.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL**: status

**HAL\_LTDC\_EnableCLUT**
**Function name**

HAL\_StatusTypeDef HAL\_LTDC\_EnableCLUT (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)

**Function description**

Enable the color lookup table.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL**: status



### HAL\_LTDC\_DisableCLUT

#### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_DisableCLUT (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)**

#### Function description

Disable the color lookup table.

#### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

#### Return values

- **HAL**: status

### HAL\_LTDC\_ProgramLineEvent

#### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_ProgramLineEvent (LTDC\_HandleTypeDef \* hltdc, uint32\_t Line)**

#### Function description

Define the position of the line interrupt.

#### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Line**: Line Interrupt Position.

#### Return values

- **HAL**: status

#### Notes

- User application may resort to HAL\_LTDC\_LineEventCallback() at line interrupt generation.

### HAL\_LTDC\_EnableDither

#### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_EnableDither (LTDC\_HandleTypeDef \* hltdc)**

#### Function description

Enable Dither.

#### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

#### Return values

- **HAL**: status

### HAL\_LTDC\_DisableDither

#### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_DisableDither (LTDC\_HandleTypeDef \* hltdc)**

#### Function description

Disable Dither.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

**Return values**

- **HAL**: status

**HAL\_LTDC\_Reload**
**Function name**

**HAL\_StatusTypeDef HAL\_LTDC\_Reload (LTDC\_HandleTypeDef \* hltdc, uint32\_t ReloadType)**

**Function description**

Reload LTDC Layers configuration.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **ReloadType**: This parameter can be one of the following values : LTDC\_RELOAD\_IMMEDIATE : Immediate Reload LTDC\_RELOAD\_VERTICAL\_BLANKING : Reload in the next Vertical Blanking

**Return values**

- **HAL**: status

**Notes**

- User application may resort to HAL\_LTDC\_ReloadEventCallback() at reload interrupt generation.

**HAL\_LTDC\_ConfigLayer\_NoReload**
**Function name**

**HAL\_StatusTypeDef HAL\_LTDC\_ConfigLayer\_NoReload (LTDC\_HandleTypeDef \* hltdc, LTDC\_LayerCfgTypeDef \* pLayerCfg, uint32\_t LayerIdx)**

**Function description**

Configure the LTDC Layer according to the specified without reloading parameters in the LTDC\_InitTypeDef and create the associated handle.

**Parameters**

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg**: pointer to a LTDC\_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL**: status

**HAL\_LTDC\_SetWindowSize\_NoReload**
**Function name**

**HAL\_StatusTypeDef HAL\_LTDC\_SetWindowSize\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t XSize, uint32\_t YSize, uint32\_t LayerIdx)**

**Function description**

Set the LTDC window size without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize**: LTDC Pixel per line
- **YSize**: LTDC Line number
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

#### HAL\_LTDC\_SetWindowPosition\_NoReload

### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_SetWindowPosition\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t X0, uint32\_t Y0, uint32\_t LayerIdx)**

### Function description

Set the LTDC window position without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0**: LTDC window X offset
- **Y0**: LTDC window Y offset
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

#### HAL\_LTDC\_SetPixelFormat\_NoReload

### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_SetPixelFormat\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t Pixelformat, uint32\_t LayerIdx)**

### Function description

Reconfigure the pixel format without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **PixelFormat**: new pixel format value.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1).

### Return values

- **HAL**: status

#### HAL\_LTDC\_SetAlpha\_NoReload

### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_SetAlpha\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t Alpha, uint32\_t LayerIdx)**

### Function description

Reconfigure the layer alpha value without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha**: new alpha value.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

#### HAL\_LTDC\_SetAddress\_NoReload

### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_SetAddress\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t Address, uint32\_t LayerIdx)**

### Function description

Reconfigure the frame buffer Address without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address**: new address value.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1).

### Return values

- **HAL**: status

#### HAL\_LTDC\_SetPitch\_NoReload

### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_SetPitch\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t LinePitchInPixels, uint32\_t LayerIdx)**

### Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels**: New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx**: LTDC layer index concerned by the modification of line pitch.

### Return values

- **HAL**: status

### Notes

- This function should be called only after a previous call to HAL\_LTDC\_ConfigLayer() to modify the default pitch configured by HAL\_LTDC\_ConfigLayer() when required (refer to example described just above). Variant of the function HAL\_LTDC\_SetPitch without immediate reload.

#### HAL\_LTDC\_ConfigColorKeying\_NoReload

### Function name

**HAL\_StatusTypeDef HAL\_LTDC\_ConfigColorKeying\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t RGBValue, uint32\_t LayerIdx)**

### Function description

Configure the color keying without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue**: the color key value
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_EnableColorKeying\_NoReload

### Function name

HAL\_StatusTypeDef HAL\_LTDC\_EnableColorKeying\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)

### Function description

Enable the color keying without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_DisableColorKeying\_NoReload

### Function name

HAL\_StatusTypeDef HAL\_LTDC\_DisableColorKeying\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)

### Function description

Disable the color keying without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

### Return values

- **HAL**: status

### HAL\_LTDC\_EnableCLUT\_NoReload

### Function name

HAL\_StatusTypeDef HAL\_LTDC\_EnableCLUT\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)

### Function description

Enable the color lookup table without reloading.

### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx**: LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL:** status

**HAL\_LTDC\_DisableCLUT\_NoReload**

**Function name**

**HAL\_StatusTypeDef HAL\_LTDC\_DisableCLUT\_NoReload (LTDC\_HandleTypeDef \* hltdc, uint32\_t LayerIdx)**

**Function description**

Disable the color lookup table without reloading.

**Parameters**

- **hltdc:** pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC\_LAYER\_1 (0) or LTDC\_LAYER\_2 (1)

**Return values**

- **HAL:** status

**HAL\_LTDC\_GetState**

**Function name**

**HAL\_LTDC\_StateTypeDef HAL\_LTDC\_GetState (LTDC\_HandleTypeDef \* hltdc)**

**Function description**

Return the LTDC handle state.

**Parameters**

- **hltdc:** pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

**Return values**

- **HAL:** state

**HAL\_LTDC\_GetError**

**Function name**

**uint32\_t HAL\_LTDC\_GetError (LTDC\_HandleTypeDef \* hltdc)**

**Function description**

Return the LTDC handle error code.

**Parameters**

- **hltdc:** pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.

**Return values**

- **LTDC:** Error Code

## 43.3 LTDC Firmware driver defines

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

### 43.3.1 LTDC

LTDC

*LTDC Alpha*

#### LTDC\_ALPHA

LTDC Constant Alpha mask

**LTDC BACK COLOR**

**LTDC\_COLOR**

Color mask

**LTDC Blending Factor1**

**LTDC\_BLENDING\_FACTOR1\_CA**

Blending factor : Cte Alpha

**LTDC\_BLENDING\_FACTOR1\_PAxCA**

Blending factor : Cte Alpha x Pixel Alpha

**LTDC Blending Factor2**

**LTDC\_BLENDING\_FACTOR2\_CA**

Blending factor : Cte Alpha

**LTDC\_BLENDING\_FACTOR2\_PAxCA**

Blending factor : Cte Alpha x Pixel Alpha

**LTDC DE POLARITY**

**LTDC\_DEPOLARITY\_AL**

Data Enable, is active low.

**LTDC\_DEPOLARITY\_AH**

Data Enable, is active high.

**LTDC Error Code**

**HAL\_LTDC\_ERROR\_NONE**

LTDC No error

**HAL\_LTDC\_ERROR\_TE**

LTDC Transfer error

**HAL\_LTDC\_ERROR\_FU**

LTDC FIFO Underrun

**HAL\_LTDC\_ERROR\_TIMEOUT**

LTDC Timeout error

**LTDC Exported Macros**

**\_\_HAL\_LTDC\_RESET\_HANDLE\_STATE**

**Description:**

- Reset LTDC handle state.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None

**\_\_HAL\_LTDC\_ENABLE**

**Description:**

- Enable the LTDC.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.

### `__HAL_LTDC_DISABLE`

**Description:**

- Disable the LTDC.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.

### `__HAL_LTDC_LAYER_ENABLE`

**Description:**

- Enable the LTDC Layer.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__LAYER__`: Specify the layer to be enabled. This parameter can be `LTDC_LAYER_1` (0) or `LTDC_LAYER_2` (1).

**Return value:**

- None.

### `__HAL_LTDC_LAYER_DISABLE`

**Description:**

- Disable the LTDC Layer.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__LAYER__`: Specify the layer to be disabled. This parameter can be `LTDC_LAYER_1` (0) or `LTDC_LAYER_2` (1).

**Return value:**

- None.

### `__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG`

**Description:**

- Reload immediately all LTDC Layers.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.

### `__HAL_LTDC_VERTICAL_BLANKING_RELOAD_CONFIG`

**Description:**

- Reload during vertical blanking period all LTDC Layers.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.



### `__HAL_LTDC_GET_FLAG`

**Description:**

- Get the LTDC pending flags.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
  - `LTDC_FLAG_LI`: Line Interrupt flag
  - `LTDC_FLAG_FU`: FIFO Underrun Interrupt flag
  - `LTDC_FLAG_TE`: Transfer Error interrupt flag
  - `LTDC_FLAG_RR`: Register Reload Interrupt Flag

**Return value:**

- The: state of FLAG (SET or RESET).

### `__HAL_LTDC_CLEAR_FLAG`

**Description:**

- Clears the LTDC pending flags.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__FLAG__`: Specify the flag to clear. This parameter can be any combination of the following values:
  - `LTDC_FLAG_LI`: Line Interrupt flag
  - `LTDC_FLAG_FU`: FIFO Underrun Interrupt flag
  - `LTDC_FLAG_TE`: Transfer Error interrupt flag
  - `LTDC_FLAG_RR`: Register Reload Interrupt Flag

**Return value:**

- None

### `__HAL_LTDC_ENABLE_IT`

**Description:**

- Enables the specified LTDC interrupts.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__INTERRUPT__`: Specify the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - `LTDC_IT_LI`: Line Interrupt flag
  - `LTDC_IT_FU`: FIFO Underrun Interrupt flag
  - `LTDC_IT_TE`: Transfer Error interrupt flag
  - `LTDC_IT_RR`: Register Reload Interrupt Flag

**Return value:**

- None

### **\_\_HAL\_LTDC\_DISABLE\_IT**

**Description:**

- Disables the specified LTDC interrupts.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__INTERRUPT__`: Specify the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - `LTDC_IT_LI`: Line Interrupt flag
  - `LTDC_IT_FU`: FIFO Underrun Interrupt flag
  - `LTDC_IT_TE`: Transfer Error interrupt flag
  - `LTDC_IT_RR`: Register Reload Interrupt Flag

**Return value:**

- None

### **\_\_HAL\_LTDC\_GET\_IT\_SOURCE**

**Description:**

- Check whether the specified LTDC interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__INTERRUPT__`: Specify the LTDC interrupt source to check. This parameter can be one of the following values:
  - `LTDC_IT_LI`: Line Interrupt flag
  - `LTDC_IT_FU`: FIFO Underrun Interrupt flag
  - `LTDC_IT_TE`: Transfer Error interrupt flag
  - `LTDC_IT_RR`: Register Reload Interrupt Flag

**Return value:**

- The: state of INTERRUPT (SET or RESET).

**LTDC Exported Types**

### **MAX\_LAYER**

**LTDC Flags**

#### **LTDC\_FLAG\_LI**

LTDC Line Interrupt Flag

#### **LTDC\_FLAG\_FU**

LTDC FIFO Underrun interrupt Flag

#### **LTDC\_FLAG\_TE**

LTDC Transfer Error interrupt Flag

#### **LTDC\_FLAG\_RR**

LTDC Register Reload interrupt Flag

**LTDC HS POLARITY**

#### **LTDC\_HSPOLARITY\_AL**

Horizontal Synchronization is active low.

#### **LTDC\_HSPOLARITY\_AH**

Horizontal Synchronization is active high.

**LTDC Interrupts**

**LTDC\_IT\_LI**

LTDC Line Interrupt

**LTDC\_IT\_FU**

LTDC FIFO Underrun Interrupt

**LTDC\_IT\_TE**

LTDC Transfer Error Interrupt

**LTDC\_IT\_RR**

LTDC Register Reload Interrupt

***LTDC Layer*****LTDC\_LAYER\_1**

LTDC Layer 1

**LTDC\_LAYER\_2**

LTDC Layer 2

***LTDC LAYER Config*****LTDC\_STOPPOSITION**

LTDC Layer stop position

**LTDC\_STARTPOSITION**

LTDC Layer start position

**LTDC\_COLOR\_FRAME\_BUFFER**

LTDC Layer Line length

**LTDC\_LINE\_NUMBER**

LTDC Layer Line number

***LTDC PC POLARITY*****LTDC\_PCPOLARITY\_IPC**

input pixel clock.

**LTDC\_PCPOLARITY\_IIPC**

inverted input pixel clock.

***LTDC Pixel format*****LTDC\_PIXEL\_FORMAT\_ARGB8888**

ARGB8888 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_RGB888**

RGB888 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_RGB565**

RGB565 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_ARGB1555**

ARGB1555 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_ARGB4444**

ARGB4444 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_L8**

L8 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_AL44**

AL44 LTDC pixel format

**LTDC\_PIXEL\_FORMAT\_AL88**

AL88 LTDC pixel format

***LTDC Reload Type***

**LTDC\_RELOAD\_IMMEDIATE**

Immediate Reload

**LTDC\_RELOAD\_VERTICAL\_BLANKING**

Vertical Blanking Reload

***LTDC SYNC***

**LTDC\_HORIZONTALSYNC**

Horizontal synchronization width.

**LTDC\_VERTICALSYNC**

Vertical synchronization height.

***LTDC VS POLARITY***

**LTDC\_VSPOLARITY\_AL**

Vertical Synchronization is active low.

**LTDC\_VSPOLARITY\_AH**

Vertical Synchronization is active high.

## 44 HAL LTDC Extension Driver

### 44.1 LTDCEx Firmware driver API description

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

#### 44.1.1 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC

This section contains the following APIs:

- [HAL\\_LTDCEx\\_StructInitFromVideoConfig\(\)](#)
- [HAL\\_LTDCEx\\_StructInitFromAdaptedCommandConfig\(\)](#)

#### 44.1.2 Detailed description of functions

##### HAL\_LTDCEx\_StructInitFromVideoConfig

###### Function name

**HAL\_StatusTypeDef HAL\_LTDCEx\_StructInitFromVideoConfig (LTDC\_HandleTypeDef \* hltdc, DSI\_VidCfgTypeDef \* VidCfg)**

###### Function description

Retrieve common parameters from DSI Video mode configuration structure.

###### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **VidCfg**: pointer to a DSI\_VidCfgTypeDef structure that contains the DSI video mode configuration parameters

###### Return values

- **HAL**: status

###### Notes

- The implementation of this function is taking into account the LTDC polarities inversion as described in the current LTDC specification

##### HAL\_LTDCEx\_StructInitFromAdaptedCommandConfig

###### Function name

**HAL\_StatusTypeDef HAL\_LTDCEx\_StructInitFromAdaptedCommandConfig (LTDC\_HandleTypeDef \* hltdc, DSI\_CmdCfgTypeDef \* CmdCfg)**

###### Function description

Retrieve common parameters from DSI Adapted command mode configuration structure.

###### Parameters

- **hltdc**: pointer to a LTDC\_HandleTypeDef structure that contains the configuration information for the LTDC.
- **CmdCfg**: pointer to a DSI\_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters

###### Return values

- **HAL**: status

###### Notes

- The implementation of this function is taking into account the LTDC polarities inversion as described in the current LTDC specification

## 45 HAL MMC Generic Driver

### 45.1 MMC Firmware driver registers structures

#### 45.1.1 HAL\_MMC\_CardInfoTypeDef

*HAL\_MMC\_CardInfoTypeDef* is defined in the `stm32f4xx_hal_mmc.h`

##### Data Fields

- *uint32\_t CardType*
- *uint32\_t Class*
- *uint32\_t RelCardAdd*
- *uint32\_t BlockNbr*
- *uint32\_t BlockSize*
- *uint32\_t LogBlockNbr*
- *uint32\_t LogBlockSize*

##### Field Documentation

- *uint32\_t HAL\_MMC\_CardInfoTypeDef::CardType*  
Specifies the card Type
- *uint32\_t HAL\_MMC\_CardInfoTypeDef::Class*  
Specifies the class of the card class
- *uint32\_t HAL\_MMC\_CardInfoTypeDef::RelCardAdd*  
Specifies the Relative Card Address
- *uint32\_t HAL\_MMC\_CardInfoTypeDef::BlockNbr*  
Specifies the Card Capacity in blocks
- *uint32\_t HAL\_MMC\_CardInfoTypeDef::BlockSize*  
Specifies one block size in bytes
- *uint32\_t HAL\_MMC\_CardInfoTypeDef::LogBlockNbr*  
Specifies the Card logical Capacity in blocks
- *uint32\_t HAL\_MMC\_CardInfoTypeDef::LogBlockSize*  
Specifies logical block size in bytes

#### 45.1.2 MMC\_HandleTypeDef

*MMC\_HandleTypeDef* is defined in the `stm32f4xx_hal_mmc.h`

##### Data Fields

- *MMC\_TypeDef \* Instance*
- *MMC\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *uint8\_t \* pTxBuffPtr*
- *uint32\_t TxXferSize*
- *uint8\_t \* pRxBuffPtr*
- *uint32\_t RxXferSize*
- *\_\_IO uint32\_t Context*
- *\_\_IO HAL\_MMC\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*
- *DMA\_HandleTypeDef \* hdmarx*
- *DMA\_HandleTypeDef \* hdmatx*
- *HAL\_MMC\_CardInfoTypeDef MmcCard*
- *uint32\_t CSD*
- *uint32\_t CID*

### Field Documentation

- ***MMC\_TypeDef\* MMC\_HandleTypeDef::Instance***  
MMC registers base address
- ***MMC\_InitTypeDef MMC\_HandleTypeDef::Init***  
MMC required parameters
- ***HAL\_LockTypeDef MMC\_HandleTypeDef::Lock***  
MMC locking object
- ***uint8\_t\* MMC\_HandleTypeDef::pTxBuffPtr***  
Pointer to MMC Tx transfer Buffer
- ***uint32\_t MMC\_HandleTypeDef::TxXferSize***  
MMC Tx Transfer size
- ***uint8\_t\* MMC\_HandleTypeDef::pRxBuffPtr***  
Pointer to MMC Rx transfer Buffer
- ***uint32\_t MMC\_HandleTypeDef::RxXferSize***  
MMC Rx Transfer size
- ***\_\_IO uint32\_t MMC\_HandleTypeDef::Context***  
MMC transfer context
- ***\_\_IO HAL\_MMC\_StateTypeDef MMC\_HandleTypeDef::State***  
MMC card State
- ***\_\_IO uint32\_t MMC\_HandleTypeDef::ErrorCode***  
MMC Card Error codes
- ***DMA\_HandleTypeDef\* MMC\_HandleTypeDef::hdmarx***  
MMC Rx DMA handle parameters
- ***DMA\_HandleTypeDef\* MMC\_HandleTypeDef::hdmatx***  
MMC Tx DMA handle parameters
- ***HAL\_MMC\_CardInfoTypeDef MMC\_HandleTypeDef::MmcCard***  
MMC Card information
- ***uint32\_t MMC\_HandleTypeDef::CSD[4U]***  
MMC card specific data table
- ***uint32\_t MMC\_HandleTypeDef::CID[4U]***  
MMC card identification number table

### 45.1.3 HAL\_MMC\_CardCSDTypeDef

***HAL\_MMC\_CardCSDTypeDef*** is defined in the `stm32f4xx_hal_mmc.h`

#### Data Fields

- ***\_\_IO uint8\_t CSDStruct***
- ***\_\_IO uint8\_t SysSpecVersion***
- ***\_\_IO uint8\_t Reserved1***
- ***\_\_IO uint8\_t TAAC***
- ***\_\_IO uint8\_t NSAC***
- ***\_\_IO uint8\_t MaxBusClkFrec***
- ***\_\_IO uint16\_t CardComdClasses***
- ***\_\_IO uint8\_t RdBlockLen***
- ***\_\_IO uint8\_t PartBlockRead***
- ***\_\_IO uint8\_t WrBlockMisalign***
- ***\_\_IO uint8\_t RdBlockMisalign***
- ***\_\_IO uint8\_t DSRImpl***
- ***\_\_IO uint8\_t Reserved2***
- ***\_\_IO uint32\_t DeviceSize***
- ***\_\_IO uint8\_t MaxRdCurrentVDDMin***

- `__IO uint8_t MaxRdCurrentVDDMax`
- `__IO uint8_t MaxWrCurrentVDDMin`
- `__IO uint8_t MaxWrCurrentVDDMax`
- `__IO uint8_t DeviceSizeMul`
- `__IO uint8_t EraseGrSize`
- `__IO uint8_t EraseGrMul`
- `__IO uint8_t WrProtectGrSize`
- `__IO uint8_t WrProtectGrEnable`
- `__IO uint8_t ManDefIECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGroup`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

#### Field Documentation

- `__IO uint8_t HAL_MMC_CardCSDTypeDef::CSDStruct`  
CSD structure
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::SysSpecVersion`  
System specification version
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved1`  
Reserved
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::TAAC`  
Data read access time 1
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::NSAC`  
Data read access time 2 in CLK cycles
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxBusClkFrec`  
Max. bus clock frequency
- `__IO uint16_t HAL_MMC_CardCSDTypeDef::CardComdClasses`  
Card command classes
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockLen`  
Max. read data block length
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::PartBlockRead`  
Partial blocks for read allowed
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::WrBlockMisalign`  
Write block misalignment
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockMisalign`  
Read block misalignment
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::DSRImpl`  
DSR implemented
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved2`  
Reserved



- **\_\_IO uint32\_t HAL\_MMC\_CardCSDTypeDef::DeviceSize**  
Device Size
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::MaxRdCurrentVDDMin**  
Max. read current @ VDD min
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::MaxRdCurrentVDDMax**  
Max. read current @ VDD max
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::MaxWrCurrentVDDMin**  
Max. write current @ VDD min
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::MaxWrCurrentVDDMax**  
Max. write current @ VDD max
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::DeviceSizeMul**  
Device size multiplier
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::EraseGrSize**  
Erase group size
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::EraseGrMul**  
Erase group size multiplier
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::WrProtectGrSize**  
Write protect group size
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::WrProtectGrEnable**  
Write protect group enable
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::ManDefIECC**  
Manufacturer default ECC
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::WrSpeedFact**  
Write speed factor
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::MaxWrBlockLen**  
Max. write data block length
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::WriteBlockPaPartial**  
Partial blocks for write allowed
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::Reserved3**  
Reserved
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::ContentProtectAppli**  
Content protection application
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::FileFormatGroup**  
File format group
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::CopyFlag**  
Copy flag (OTP)
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::PermWrProtect**  
Permanent write protection
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::TempWrProtect**  
Temporary write protection
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::FileFormat**  
File format
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::ECC**  
ECC code
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::CSD\_CRC**  
CSD CRC
- **\_\_IO uint8\_t HAL\_MMC\_CardCSDTypeDef::Reserved4**  
Always 1

#### 45.1.4 HAL\_MMC\_CardCIDTypeDef

*HAL\_MMC\_CardCIDTypeDef* is defined in the stm32f4xx\_hal\_mmc.h

##### Data Fields

- *\_\_IO uint8\_t ManufacturerID*
- *\_\_IO uint16\_t OEM\_AppliID*
- *\_\_IO uint32\_t ProdName1*
- *\_\_IO uint8\_t ProdName2*
- *\_\_IO uint8\_t ProdRev*
- *\_\_IO uint32\_t ProdSN*
- *\_\_IO uint8\_t Reserved1*
- *\_\_IO uint16\_t ManufactDate*
- *\_\_IO uint8\_t CID\_CRC*
- *\_\_IO uint8\_t Reserved2*

##### Field Documentation

- *\_\_IO uint8\_t HAL\_MMC\_CardCIDTypeDef::ManufacturerID*  
Manufacturer ID
- *\_\_IO uint16\_t HAL\_MMC\_CardCIDTypeDef::OEM\_AppliID*  
OEM/Application ID
- *\_\_IO uint32\_t HAL\_MMC\_CardCIDTypeDef::ProdName1*  
Product Name part1
- *\_\_IO uint8\_t HAL\_MMC\_CardCIDTypeDef::ProdName2*  
Product Name part2
- *\_\_IO uint8\_t HAL\_MMC\_CardCIDTypeDef::ProdRev*  
Product Revision
- *\_\_IO uint32\_t HAL\_MMC\_CardCIDTypeDef::ProdSN*  
Product Serial Number
- *\_\_IO uint8\_t HAL\_MMC\_CardCIDTypeDef::Reserved1*  
Reserved1
- *\_\_IO uint16\_t HAL\_MMC\_CardCIDTypeDef::ManufactDate*  
Manufacturing Date
- *\_\_IO uint8\_t HAL\_MMC\_CardCIDTypeDef::CID\_CRC*  
CID CRC
- *\_\_IO uint8\_t HAL\_MMC\_CardCIDTypeDef::Reserved2*  
Always 1

## 45.2 MMC Firmware driver API description

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

### 45.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL\_MMC\_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with MMC and eMMC cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the HAL\_MMC\_MspInit() API:
  - a. Enable the SDMMC interface clock using `__HAL_RCC_SDMMC_CLK_ENABLE()`;
  - b. SDMMC pins configuration for MMC card
    - Enable the clock for the SDMMC GPIOs using the functions `__HAL_RCC_GPIOx_CLK_ENABLE()`;
    - Configure these SDMMC pins as alternate function pull-up using `HAL_GPIO_Init()` and according to your pin assignment;
  - c. DMA Configuration if you need to use DMA process (`HAL_MMC_ReadBlocks_DMA()` and `HAL_MMC_WriteBlocks_DMA()` APIs).
    - Enable the DMAx interface clock using `__HAL_RCC_DMAx_CLK_ENABLE()`;
    - Configure the DMA using the function `HAL_DMA_Init()` with predeclared and filled.
  - d. NVIC configuration if you need to use interrupt process when using DMA transfer.
    - Configure the SDMMC and DMA interrupt priorities using function `HAL_NVIC_SetPriority()`; DMA priority is superior to SDMMC's priority
    - Enable the NVIC DMA and SDMMC IRQs using function `HAL_NVIC_EnableIRQ()`
    - SDMMC interrupts are managed using the macros `__HAL_MMC_ENABLE_IT()` and `__HAL_MMC_DISABLE_IT()` inside the communication process.
    - SDMMC interrupts pending bits are managed using the macros `__HAL_MMC_GET_IT()` and `__HAL_MMC_CLEAR_IT()`
  - e. NVIC configuration if you need to use interrupt process (`HAL_MMC_ReadBlocks_IT()` and `HAL_MMC_WriteBlocks_IT()` APIs).
    - Configure the SDMMC interrupt priorities using function `HAL_NVIC_SetPriority()`;
    - Enable the NVIC SDMMC IRQs using function `HAL_NVIC_EnableIRQ()`
    - SDMMC interrupts are managed using the macros `__HAL_MMC_ENABLE_IT()` and `__HAL_MMC_DISABLE_IT()` inside the communication process.
    - SDMMC interrupts pending bits are managed using the macros `__HAL_MMC_GET_IT()` and `__HAL_MMC_CLEAR_IT()`
2. At this stage, you can perform MMC read/write/erase operations after MMC card initialization

### MMC Card Initialization and configuration

To initialize the MMC Card, use the `HAL_MMC_Init()` function. It Initializes SDMMC Peripheral (STM32 side) and the MMC Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDMMC peripheral interface with default configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The MMC Card frequency (`SDMMC_CK`) is computed as follows:  $SDMMC\_CK = SDMMCCLK / (ClockDiv + 2)$ . In initialization mode and according to the MMC Card standard, make sure that the `SDMMC_CK` frequency doesn't exceed 400KHz. This phase of initialization is done through `SDMMC_Init()` and `SDMMC_PowerState_ON()` SDMMC low level APIs.
2. Initialize the MMC card. The API used is `HAL_MMC_InitCard()`. This phase allows the card initialization and identification and check the MMC Card type (Standard Capacity or High Capacity) The initialization flow is compatible with MMC standard. This API (`HAL_MMC_InitCard()`) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the MMC Card Data transfer frequency. By Default, the card transfer frequency is set to 24MHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the MMC Card standard, make sure that the `SDMMC_CK` frequency doesn't exceed 25MHz and 50MHz in High-speed mode switch. To be able to use a frequency higher than 24MHz, you should use the SDMMC peripheral in bypass mode. Refer to the corresponding reference manual for more details.
4. Select the corresponding MMC Card according to the address read with the step 2.
5. Configure the MMC Card in wide bus mode: 4-bits data.

### MMC Card Read operation

- You can read from MMC card in polling mode by using function `HAL_MMC_ReadBlocks()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_MMC_GetCardState()` function for MMC card state.
- You can read from MMC card in DMA mode by using function `HAL_MMC_ReadBlocks_DMA()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_MMC_GetCardState()` function for MMC card state. You could also check the DMA transfer process through the MMC Rx interrupt event.
- You can read from MMC card in Interrupt mode by using function `HAL_MMC_ReadBlocks_IT()`. This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_MMC_GetCardState()` function for MMC card state. You could also check the IT transfer process through the MMC Rx interrupt event.

### MMC Card Write operation

- You can write to MMC card in polling mode by using function `HAL_MMC_WriteBlocks()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_MMC_GetCardState()` function for MMC card state.
- You can write to MMC card in DMA mode by using function `HAL_MMC_WriteBlocks_DMA()`. This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_MMC_GetCardState()` function for MMC card state. You could also check the DMA transfer process through the MMC Tx interrupt event.
- You can write to MMC card in Interrupt mode by using function `HAL_MMC_WriteBlocks_IT()`. This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_MMC_GetCardState()` function for MMC card state. You could also check the IT transfer process through the MMC Tx interrupt event.

### MMC card information

- To get MMC card information, you can use the function `HAL_MMC_GetCardInfo()`. It returns useful information about the MMC card such as block size, card type, block number ...

### MMC card CSD register

- The `HAL_MMC_GetCardCSD()` API allows to get the parameters of the CSD register. Some of the CSD parameters are useful for card initialization and identification.

### MMC card CID register

- The `HAL_MMC_GetCardCID()` API allows to get the parameters of the CID register. Some of the CID parameters are useful for card initialization and identification.

### MMC HAL driver macros list

Below the list of most used macros in MMC HAL driver.

- `__HAL_MMC_ENABLE` : Enable the MMC device
- `__HAL_MMC_DISABLE` : Disable the MMC device
- `__HAL_MMC_DMA_ENABLE`: Enable the SDMMC DMA transfer
- `__HAL_MMC_DMA_DISABLE`: Disable the SDMMC DMA transfer
- `__HAL_MMC_ENABLE_IT`: Enable the MMC device interrupt

- `__HAL_MMC_DISABLE_IT`: Disable the MMC device interrupt
- `__HAL_MMC_GET_FLAG`: Check whether the specified MMC flag is set or not
- `__HAL_MMC_CLEAR_FLAG`: Clear the MMC's pending flags

*Note:* You can refer to the MMC HAL driver header file for more useful macros

### Callback registration

The compilation define `USE_HAL_MMC_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions `@ref HAL_MMC_RegisterCallback()` to register a user callback, it allows to register following callbacks:

- `TxCpltCallback` : callback when a transmission transfer is completed.
- `RxCpltCallback` : callback when a reception transfer is completed.
- `ErrorCallback` : callback when error occurs.
- `AbortCpltCallback` : callback when abort is completed.
- `MspInitCallback` : MMC `MspInit`.
- `MspDeInitCallback` : MMC `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_MMC_UnRegisterCallback()` to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- `TxCpltCallback` : callback when a transmission transfer is completed.
- `RxCpltCallback` : callback when a reception transfer is completed.
- `ErrorCallback` : callback when error occurs.
- `AbortCpltCallback` : callback when abort is completed.
- `MspInitCallback` : MMC `MspInit`.
- `MspDeInitCallback` : MMC `MspDeInit`. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the `@ref HAL_MMC_Init` and if the state is `HAL_MMC_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_MMC_Init` and `@ref HAL_MMC_DeInit` only when these callbacks are null (not registered beforehand). If not, `MspInit` or `MspDeInit` are not null, the `@ref HAL_MMC_Init` and `@ref HAL_MMC_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_MMC_RegisterCallback` before calling `@ref HAL_MMC_DeInit` or `@ref HAL_MMC_Init` function. When The compilation define `USE_HAL_MMC_REGISTER_CALLBACKS` is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

## 45.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the MMC card device to be ready for use.

This section contains the following APIs:

- `HAL_MMC_Init()`
- `HAL_MMC_InitCard()`
- `HAL_MMC_DeInit()`
- `HAL_MMC_MspInit()`
- `HAL_MMC_MspDeInit()`

## 45.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to MMC card.

This section contains the following APIs:

- `HAL_MMC_ReadBlocks()`
- `HAL_MMC_WriteBlocks()`
- `HAL_MMC_ReadBlocks_IT()`
- `HAL_MMC_WriteBlocks_IT()`
- `HAL_MMC_ReadBlocks_DMA()`

- *HAL\_MMC\_WriteBlocks\_DMA()*
- *HAL\_MMC\_Erase()*
- *HAL\_MMC\_IRQHandler()*
- *HAL\_MMC\_GetState()*
- *HAL\_MMC\_GetError()*
- *HAL\_MMC\_TxCpltCallback()*
- *HAL\_MMC\_RxCpltCallback()*
- *HAL\_MMC\_ErrorCallback()*
- *HAL\_MMC\_AbortCallback()*

#### 45.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MMC card operations and get the related information

This section contains the following APIs:

- *HAL\_MMC\_GetCardCID()*
- *HAL\_MMC\_GetCardCSD()*
- *HAL\_MMC\_GetCardInfo()*
- *HAL\_MMC\_ConfigWideBusOperation()*
- *HAL\_MMC\_GetCardState()*
- *HAL\_MMC\_Abort()*
- *HAL\_MMC\_Abort\_IT()*

#### 45.2.5 Detailed description of functions

##### HAL\_MMC\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_MMC\_Init (MMC\_HandleTypeDef \* hmmc)**

###### Function description

Initializes the MMC according to the specified parameters in the MMC\_HandleTypeDef and create the associated handle.

###### Parameters

- **hmmc**: Pointer to the MMC handle

###### Return values

- **HAL**: status

##### HAL\_MMC\_InitCard

###### Function name

**HAL\_StatusTypeDef HAL\_MMC\_InitCard (MMC\_HandleTypeDef \* hmmc)**

###### Function description

Initializes the MMC Card.

###### Parameters

- **hmmc**: Pointer to MMC handle

###### Return values

- **HAL**: status

###### Notes

- This function initializes the MMC card. It could be used when a card re-initialization is needed.

### HAL\_MMC\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_DeInit (MMC\_HandleTypeDef \* hmmc)**

#### Function description

De-Initializes the MMC card.

#### Parameters

- **hmmc**: Pointer to MMC handle

#### Return values

- **HAL**: status

### HAL\_MMC\_Msplnit

#### Function name

**void HAL\_MMC\_Msplnit (MMC\_HandleTypeDef \* hmmc)**

#### Function description

Initializes the MMC MSP.

#### Parameters

- **hmmc**: Pointer to MMC handle

#### Return values

- **None**:

### HAL\_MMC\_MspDeInit

#### Function name

**void HAL\_MMC\_MspDeInit (MMC\_HandleTypeDef \* hmmc)**

#### Function description

De-Initialize MMC MSP.

#### Parameters

- **hmmc**: Pointer to MMC handle

#### Return values

- **None**:

### HAL\_MMC\_ReadBlocks

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_ReadBlocks (MMC\_HandleTypeDef \* hmmc, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks, uint32\_t Timeout)**

#### Function description

Reads block(s) from a specified address in a card.

#### Parameters

- **hmmc**: Pointer to MMC handle
- **pData**: pointer to the buffer that will contain the received data
- **BlockAdd**: Block Address from where data is to be read
- **NumberOfBlocks**: Number of MMC blocks to read
- **Timeout**: Specify timeout value

#### Return values

- **HAL:** status

#### Notes

- This API should be followed by a check on the card state through `HAL_MMC_GetCardState()`.

#### **HAL\_MMC\_WriteBlocks**

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_WriteBlocks (MMC\_HandleTypeDef \* hmmc, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks, uint32\_t Timeout)**

#### Function description

Allows to write block(s) to a specified address in a card.

#### Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of MMC blocks to write
- **Timeout:** Specify timeout value

#### Return values

- **HAL:** status

#### Notes

- This API should be followed by a check on the card state through `HAL_MMC_GetCardState()`.

#### **HAL\_MMC\_Erase**

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_Erase (MMC\_HandleTypeDef \* hmmc, uint32\_t BlockStartAdd, uint32\_t BlockEndAdd)**

#### Function description

Erases the specified memory area of the given MMC card.

#### Parameters

- **hmmc:** Pointer to MMC handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

#### Return values

- **HAL:** status

#### Notes

- This API should be followed by a check on the card state through `HAL_MMC_GetCardState()`.

#### **HAL\_MMC\_ReadBlocks\_IT**

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_ReadBlocks\_IT (MMC\_HandleTypeDef \* hmmc, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Reads block(s) from a specified address in a card.



### Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

### Return values

- **HAL:** status

### Notes

- This API should be followed by a check on the card state through HAL\_MMC\_GetCardState().
- You could also check the IT transfer process through the MMC Rx interrupt event.

### HAL\_MMC\_WriteBlocks\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_WriteBlocks\_IT (MMC\_HandleTypeDef \* hmmc, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Writes block(s) to a specified address in a card.

#### Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

#### Return values

- **HAL:** status

#### Notes

- This API should be followed by a check on the card state through HAL\_MMC\_GetCardState().
- You could also check the IT transfer process through the MMC Tx interrupt event.

### HAL\_MMC\_ReadBlocks\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_ReadBlocks\_DMA (MMC\_HandleTypeDef \* hmmc, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Reads block(s) from a specified address in a card.

#### Parameters

- **hmmc:** Pointer MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

#### Return values

- **HAL:** status

#### Notes

- This API should be followed by a check on the card state through HAL\_MMC\_GetCardState().
- You could also check the DMA transfer process through the MMC Rx interrupt event.

### HAL\_MMC\_WriteBlocks\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_WriteBlocks\_DMA (MMC\_HandleTypeDef \* hmmc, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Writes block(s) to a specified address in a card.

#### Parameters

- **hmmc**: Pointer to MMC handle
- **pData**: Pointer to the buffer that will contain the data to transmit
- **BlockAdd**: Block Address where data will be written
- **NumberOfBlocks**: Number of blocks to write

#### Return values

- **HAL**: status

#### Notes

- This API should be followed by a check on the card state through HAL\_MMC\_GetCardState().
- You could also check the DMA transfer process through the MMC Tx interrupt event.

### HAL\_MMC\_IRQHandler

#### Function name

**void HAL\_MMC\_IRQHandler (MMC\_HandleTypeDef \* hmmc)**

#### Function description

This function handles MMC card interrupt request.

#### Parameters

- **hmmc**: Pointer to MMC handle

#### Return values

- **None**:

### HAL\_MMC\_TxCpltCallback

#### Function name

**void HAL\_MMC\_TxCpltCallback (MMC\_HandleTypeDef \* hmmc)**

#### Function description

Tx Transfer completed callbacks.

#### Parameters

- **hmmc**: Pointer to MMC handle

#### Return values

- **None**:

### HAL\_MMC\_RxCpltCallback

#### Function name

**void HAL\_MMC\_RxCpltCallback (MMC\_HandleTypeDef \* hmmc)**

#### Function description

Rx Transfer completed callbacks.

**Parameters**

- **hmmc:** Pointer MMC handle

**Return values**

- **None:**

**HAL\_MMC\_ErrorCallback**
**Function name**

```
void HAL_MMC_ErrorCallback (MMC_HandleTypeDef * hhmc)
```

**Function description**

MMC error callbacks.

**Parameters**

- **hmmc:** Pointer MMC handle

**Return values**

- **None:**

**HAL\_MMC\_AbortCallback**
**Function name**

```
void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hhmc)
```

**Function description**

MMC Abort callbacks.

**Parameters**

- **hmmc:** Pointer MMC handle

**Return values**

- **None:**

**HAL\_MMC\_ConfigWideBusOperation**
**Function name**

```
HAL_StatusTypeDef HAL_MMC_ConfigWideBusOperation (MMC_HandleTypeDef * hhmc, uint32_t WideMode)
```

**Function description**

Enables wide bus operation for the requested card if supported by card.

**Parameters**

- **hmmc:** Pointer to MMC handle
- **WideMode:** Specifies the MMC card wide bus mode This parameter can be one of the following values:
  - SDIO\_BUS\_WIDE\_8B: 8-bit data transfer
  - SDIO\_BUS\_WIDE\_4B: 4-bit data transfer
  - SDIO\_BUS\_WIDE\_1B: 1-bit data transfer

**Return values**

- **HAL:** status

**HAL\_MMC\_GetCardState**
**Function name**

```
HAL_MMC_CardStateTypeDef HAL_MMC_GetCardState (MMC_HandleTypeDef * hhmc)
```

### Function description

Gets the current mmc card data state.

### Parameters

- **hmmc**: pointer to MMC handle

### Return values

- **Card**: state

### HAL\_MMC\_GetCardCID

### Function name

**HAL\_StatusTypeDef HAL\_MMC\_GetCardCID (MMC\_HandleTypeDef \* hmhc, HAL\_MMC\_CardCIDTypeDef \* pCID)**

### Function description

Returns information the information of the card which are stored on the CID register.

### Parameters

- **hmmc**: Pointer to MMC handle
- **pCID**: Pointer to a HAL\_MMC\_CIDTypeDef structure that contains all CID register parameters

### Return values

- **HAL**: status

### HAL\_MMC\_GetCardCSD

### Function name

**HAL\_StatusTypeDef HAL\_MMC\_GetCardCSD (MMC\_HandleTypeDef \* hmhc, HAL\_MMC\_CardCSDTypeDef \* pCSD)**

### Function description

Returns information the information of the card which are stored on the CSD register.

### Parameters

- **hmmc**: Pointer to MMC handle
- **pCSD**: Pointer to a HAL\_MMC\_CardCSDTypeDef structure that contains all CSD register parameters

### Return values

- **HAL**: status

### HAL\_MMC\_GetCardInfo

### Function name

**HAL\_StatusTypeDef HAL\_MMC\_GetCardInfo (MMC\_HandleTypeDef \* hmhc, HAL\_MMC\_CardInfoTypeDef \* pCardInfo)**

### Function description

Gets the MMC card info.

### Parameters

- **hmmc**: Pointer to MMC handle
- **pCardInfo**: Pointer to the HAL\_MMC\_CardInfoTypeDef structure that will contain the MMC card status information

### Return values

- **HAL**: status

### HAL\_MMC\_GetState

#### Function name

**HAL\_MMC\_StateTypeDef HAL\_MMC\_GetState (MMC\_HandleTypeDef \* hmmc)**

#### Function description

return the MMC state

#### Parameters

- **hmmc**: Pointer to mmc handle

#### Return values

- **HAL**: state

### HAL\_MMC\_GetError

#### Function name

**uint32\_t HAL\_MMC\_GetError (MMC\_HandleTypeDef \* hmmc)**

#### Function description

Return the MMC error code.

#### Parameters

- **hmmc**: : Pointer to a MMC\_HandleTypeDef structure that contains the configuration information.

#### Return values

- **MMC**: Error Code

### HAL\_MMC\_Abort

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_Abort (MMC\_HandleTypeDef \* hmmc)**

#### Function description

Abort the current transfer and disable the MMC.

#### Parameters

- **hmmc**: pointer to a MMC\_HandleTypeDef structure that contains the configuration information for MMC module.

#### Return values

- **HAL**: status

### HAL\_MMC\_Abort\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_MMC\_Abort\_IT (MMC\_HandleTypeDef \* hmmc)**

#### Function description

Abort the current transfer and disable the MMC (IT mode).

#### Parameters

- **hmmc**: pointer to a MMC\_HandleTypeDef structure that contains the configuration information for MMC module.

#### Return values

- **HAL**: status

## 45.3 MMC Firmware driver defines

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

### 45.3.1 MMC

MMC

***MMC Error status enumeration Structure definition***

#### HAL\_MMC\_ERROR\_NONE

No error

#### HAL\_MMC\_ERROR\_CMD\_CRC\_FAIL

Command response received (but CRC check failed)

#### HAL\_MMC\_ERROR\_DATA\_CRC\_FAIL

Data block sent/received (CRC check failed)

#### HAL\_MMC\_ERROR\_CMD\_RSP\_TIMEOUT

Command response timeout

#### HAL\_MMC\_ERROR\_DATA\_TIMEOUT

Data timeout

#### HAL\_MMC\_ERROR\_TX\_UNDERRUN

Transmit FIFO underrun

#### HAL\_MMC\_ERROR\_RX\_OVERRUN

Receive FIFO overrun

#### HAL\_MMC\_ERROR\_ADDR\_MISALIGNED

Misaligned address

#### HAL\_MMC\_ERROR\_BLOCK\_LEN\_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

#### HAL\_MMC\_ERROR\_ERASE\_SEQ\_ERR

An error in the sequence of erase command occurs

#### HAL\_MMC\_ERROR\_BAD\_ERASE\_PARAM

An invalid selection for erase groups

#### HAL\_MMC\_ERROR\_WRITE\_PROT\_VIOLATION

Attempt to program a write protect block

#### HAL\_MMC\_ERROR\_LOCK\_UNLOCK\_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

#### HAL\_MMC\_ERROR\_COM\_CRC\_FAILED

CRC check of the previous command failed

#### HAL\_MMC\_ERROR\_ILLEGAL\_CMD

Command is not legal for the card state

#### HAL\_MMC\_ERROR\_CARD\_ECC\_FAILED

Card internal ECC was applied but failed to correct the data

**HAL\_MMC\_ERROR\_CC\_ERR**

Internal card controller error

**HAL\_MMC\_ERROR\_GENERAL\_UNKNOWN\_ERR**

General or unknown error

**HAL\_MMC\_ERROR\_STREAM\_READ\_UNDERRUN**

The card could not sustain data reading in stream rmode

**HAL\_MMC\_ERROR\_STREAM\_WRITE\_OVERRUN**

The card could not sustain data programming in stream mode

**HAL\_MMC\_ERROR\_CID\_CSD\_OVERWRITE**

CID/CSD overwrite error

**HAL\_MMC\_ERROR\_WP\_ERASE\_SKIP**

Only partial address space was erased

**HAL\_MMC\_ERROR\_CARD\_ECC\_DISABLED**

Command has been executed without using internal ECC

**HAL\_MMC\_ERROR\_ERASE\_RESET**

Erase sequence was cleared before executing because an out of erase sequence command was received

**HAL\_MMC\_ERROR\_AKE\_SEQ\_ERR**

Error in sequence of authentication

**HAL\_MMC\_ERROR\_INVALID\_VOLTRANGE**

Error in case of invalid voltage range

**HAL\_MMC\_ERROR\_ADDR\_OUT\_OF\_RANGE**

Error when addressed block is out of range

**HAL\_MMC\_ERROR\_REQUEST\_NOT\_APPLICABLE**

Error when command request is not applicable

**HAL\_MMC\_ERROR\_PARAM**

the used parameter is not valid

**HAL\_MMC\_ERROR\_UNSUPPORTED\_FEATURE**

Error when feature is not insupported

**HAL\_MMC\_ERROR\_BUSY**

Error when transfer process is busy

**HAL\_MMC\_ERROR\_DMA**

Error while DMA transfer

**HAL\_MMC\_ERROR\_TIMEOUT**

Timeout error

***MMC context enumeration***

**MMC\_CONTEXT\_NONE**

None

**MMC\_CONTEXT\_READ\_SINGLE\_BLOCK**

Read single block operation

**MMC\_CONTEXT\_READ\_MULTIPLE\_BLOCK**

Read multiple blocks operation

**MMC\_CONTEXT\_WRITE\_SINGLE\_BLOCK**

Write single block operation

**MMC\_CONTEXT\_WRITE\_MULTIPLE\_BLOCK**

Write multiple blocks operation

**MMC\_CONTEXT\_IT**

Process in Interrupt mode

**MMC\_CONTEXT\_DMA**

Process in DMA mode

**MMC Voltage mode**

**MMC\_HIGH\_VOLTAGE\_RANGE**

VALUE OF ARGUMENT

**MMC\_DUAL\_VOLTAGE\_RANGE**

VALUE OF ARGUMENT

**eMMC\_HIGH\_VOLTAGE\_RANGE**

for eMMC > 2Gb sector mode

**eMMC\_DUAL\_VOLTAGE\_RANGE**

for eMMC > 2Gb sector mode

**MMC\_INVALID\_VOLTAGE\_RANGE**

**MMC Memory Cards**

**MMC\_LOW\_CAPACITY\_CARD**

MMC Card Capacity <=2Gbytes

**MMC\_HIGH\_CAPACITY\_CARD**

MMC Card Capacity >2Gbytes and <2Tbytes

**Exported Constants**

**MMC\_BLOCKSIZE**

Block size is 512 bytes

**MMC Exported Macros**

**\_\_HAL\_MMC\_RESET\_HANDLE\_STATE**

**Description:**

- Reset MMC handle state.

**Parameters:**

- `__HANDLE__`: MMC handle.

**Return value:**

- None

**\_\_HAL\_MMC\_ENABLE**

**Description:**

- Enable the MMC device.

**Return value:**

- None



**\_\_HAL\_MMC\_DISABLE**
**Description:**

- Disable the MMC device.

**Return value:**

- None

**\_\_HAL\_MMC\_DMA\_ENABLE**
**Description:**

- Enable the SDMMC DMA transfer.

**Return value:**

- None

**\_\_HAL\_MMC\_DMA\_DISABLE**
**Description:**

- Disable the SDMMC DMA transfer.

**Return value:**

- None

**\_\_HAL\_MMC\_ENABLE\_IT**
**Description:**

- Enable the MMC device interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: MMC Handle
- **\_\_INTERRUPT\_\_**: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDIO\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDIO\_IT\_DTIMEOUT: Data timeout interrupt
  - SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt
  - SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
  - SDIO\_IT\_CMDSSENT: Command sent (no response required) interrupt
  - SDIO\_IT\_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDIO\_IT\_CMDACT: Command transfer in progress interrupt
  - SDIO\_IT\_TXACT: Data transmit in progress interrupt
  - SDIO\_IT\_RXACT: Data receive in progress interrupt
  - SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDIO\_IT\_TXFIFO: Transmit FIFO full interrupt
  - SDIO\_IT\_RXFIFO: Receive FIFO full interrupt
  - SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
  - SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
  - SDIO\_IT\_TXDAVL: Data available in transmit FIFO interrupt
  - SDIO\_IT\_RXDAVL: Data available in receive FIFO interrupt
  - SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt

**Return value:**

- None

**\_\_HAL\_MMC\_DISABLE\_IT**
**Description:**

- Disable the MMC device interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: MMC Handle
- **\_\_INTERRUPT\_\_**: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - **SDIO\_IT\_CCRCFAIL**: Command response received (CRC check failed) interrupt
  - **SDIO\_IT\_DCRCFail**: Data block sent/received (CRC check failed) interrupt
  - **SDIO\_IT\_CTIMEOUT**: Command response timeout interrupt
  - **SDIO\_IT\_DTIMEOUT**: Data timeout interrupt
  - **SDIO\_IT\_TXUNDERR**: Transmit FIFO underrun error interrupt
  - **SDIO\_IT\_RXOVERR**: Received FIFO overrun error interrupt
  - **SDIO\_IT\_CMDREND**: Command response received (CRC check passed) interrupt
  - **SDIO\_IT\_CMDSSENT**: Command sent (no response required) interrupt
  - **SDIO\_IT\_DATAEND**: Data end (data counter, **DATACOUNT**, is zero) interrupt
  - **SDIO\_IT\_DBCKEND**: Data block sent/received (CRC check passed) interrupt
  - **SDIO\_IT\_CMDACT**: Command transfer in progress interrupt
  - **SDIO\_IT\_TXACT**: Data transmit in progress interrupt
  - **SDIO\_IT\_RXACT**: Data receive in progress interrupt
  - **SDIO\_IT\_TXFIFOHE**: Transmit FIFO Half Empty interrupt
  - **SDIO\_IT\_RXFIFOHF**: Receive FIFO Half Full interrupt
  - **SDIO\_IT\_TXFIFO**: Transmit FIFO full interrupt
  - **SDIO\_IT\_RXFIFO**: Receive FIFO full interrupt
  - **SDIO\_IT\_TXFIFOE**: Transmit FIFO empty interrupt
  - **SDIO\_IT\_RXFIFOE**: Receive FIFO empty interrupt
  - **SDIO\_IT\_TXDAVL**: Data available in transmit FIFO interrupt
  - **SDIO\_IT\_RXDAVL**: Data available in receive FIFO interrupt
  - **SDIO\_IT\_SDIOIT**: SD I/O interrupt received interrupt

**Return value:**

- None

**\_\_HAL\_MMC\_GET\_FLAG**
**Description:**

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

**Parameters:**

- `__HANDLE__`: MMC Handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
  - `SDIO_FLAG_DCRFAIL`: Data block sent/received (CRC check failed)
  - `SDIO_FLAG_CTIMEOUT`: Command response timeout
  - `SDIO_FLAG_DTIMEOUT`: Data timeout
  - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
  - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
  - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
  - `SDIO_FLAG_CMDSSENT`: Command sent (no response required)
  - `SDIO_FLAG_DATAEND`: Data end (data counter, `DATACOUNT`, is zero)
  - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
  - `SDIO_FLAG_CMDACT`: Command transfer in progress
  - `SDIO_FLAG_TXACT`: Data transmit in progress
  - `SDIO_FLAG_RXACT`: Data receive in progress
  - `SDIO_FLAG_TXFIFOHE`: Transmit FIFO Half Empty
  - `SDIO_FLAG_RXFIFOHF`: Receive FIFO Half Full
  - `SDIO_FLAG_TXFIFO`: Transmit FIFO full
  - `SDIO_FLAG_RXFIFO`: Receive FIFO full
  - `SDIO_FLAG_TXFIFOE`: Transmit FIFO empty
  - `SDIO_FLAG_RXFIFOE`: Receive FIFO empty
  - `SDIO_FLAG_TXDAVL`: Data available in transmit FIFO
  - `SDIO_FLAG_RXDAVL`: Data available in receive FIFO
  - `SDIO_FLAG_SDIOIT`: SD I/O interrupt received

**Return value:**

- The: new state of MMC FLAG (SET or RESET).

## \_\_HAL\_MMC\_CLEAR\_FLAG

**Description:**

- Clear the MMC's pending flags.

**Parameters:**

- `__HANDLE__`: MMC Handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
  - `SDIO_FLAG_DCRCFAIL`: Data block sent/received (CRC check failed)
  - `SDIO_FLAG_CTIMEOUT`: Command response timeout
  - `SDIO_FLAG_DTIMEOUT`: Data timeout
  - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
  - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
  - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
  - `SDIO_FLAG_CMDSSENT`: Command sent (no response required)
  - `SDIO_FLAG_DATAEND`: Data end (data counter, `DATACOUNT`, is zero)
  - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
  - `SDIO_FLAG_SDIOIT`: SD I/O interrupt received

**Return value:**

- None

**\_\_HAL\_MMC\_GET\_IT**
**Description:**

- Check whether the specified MMC interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: MMC Handle
- `__INTERRUPT__`: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRFAIL`: Data block sent/received (CRC check failed) interrupt
  - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
  - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
  - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
  - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
  - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
  - `SDIO_IT_CMDSSENT`: Command sent (no response required) interrupt
  - `SDIO_IT_DATAEND`: Data end (data counter, `DATACOUNT`, is zero) interrupt
  - `SDIO_IT_DBCKEND`: Data block sent/received (CRC check passed) interrupt
  - `SDIO_IT_CMDACT`: Command transfer in progress interrupt
  - `SDIO_IT_TXACT`: Data transmit in progress interrupt
  - `SDIO_IT_RXACT`: Data receive in progress interrupt
  - `SDIO_IT_TXFIFOHE`: Transmit FIFO Half Empty interrupt
  - `SDIO_IT_RXFIFOHF`: Receive FIFO Half Full interrupt
  - `SDIO_IT_TXFIFO`: Transmit FIFO full interrupt
  - `SDIO_IT_RXFIFO`: Receive FIFO full interrupt
  - `SDIO_IT_TXFIFOE`: Transmit FIFO empty interrupt
  - `SDIO_IT_RXFIFOE`: Receive FIFO empty interrupt
  - `SDIO_IT_TXDAVL`: Data available in transmit FIFO interrupt
  - `SDIO_IT_RXDAVL`: Data available in receive FIFO interrupt
  - `SDIO_IT_SDIOIT`: SD I/O interrupt received interrupt

**Return value:**

- The: new state of MMC IT (SET or RESET).

## **\_\_HAL\_MMC\_CLEAR\_IT**

### **Description:**

- Clear the MMC's interrupt pending bits.

### **Parameters:**

- **\_\_HANDLE\_\_**: MMC Handle
- **\_\_INTERRUPT\_\_**: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - **SDIO\_IT\_CCRCFAIL**: Command response received (CRC check failed) interrupt
  - **SDIO\_IT\_DCRFAIL**: Data block sent/received (CRC check failed) interrupt
  - **SDIO\_IT\_CTIMEOUT**: Command response timeout interrupt
  - **SDIO\_IT\_DTIMEOUT**: Data timeout interrupt
  - **SDIO\_IT\_TXUNDERR**: Transmit FIFO underrun error interrupt
  - **SDIO\_IT\_RXOVERR**: Received FIFO overrun error interrupt
  - **SDIO\_IT\_CMDREND**: Command response received (CRC check passed) interrupt
  - **SDIO\_IT\_CMDSSENT**: Command sent (no response required) interrupt
  - **SDIO\_IT\_DATAEND**: Data end (data counter, **DATACOUNT**, is zero) interrupt
  - **SDIO\_IT\_DBCKEND**: Data block sent/received (CRC check passed) interrupt
  - **SDIO\_IT\_TXFIFOHE**: Transmit FIFO Half Empty interrupt
  - **SDIO\_IT\_RXFIFOHF**: Receive FIFO Half Full interrupt
  - **SDIO\_IT\_RXFIFO**: Receive FIFO full interrupt
  - **SDIO\_IT\_TXFIFOE**: Transmit FIFO empty interrupt
  - **SDIO\_IT\_SDIOIT**: SD I/O interrupt received interrupt

### **Return value:**

- None

### **MMC Card State enumeration structure**

#### **HAL\_MMC\_CARD\_READY**

Card state is ready

#### **HAL\_MMC\_CARD\_IDENTIFICATION**

Card is in identification state

#### **HAL\_MMC\_CARD\_STANDBY**

Card is in standby state

#### **HAL\_MMC\_CARD\_TRANSFER**

Card is in transfer state

#### **HAL\_MMC\_CARD\_SENDING**

Card is sending an operation

#### **HAL\_MMC\_CARD\_RECEIVING**

Card is receiving operation information

#### **HAL\_MMC\_CARD\_PROGRAMMING**

Card is in programming state

#### **HAL\_MMC\_CARD\_DISCONNECTED**

Card is disconnected

#### **HAL\_MMC\_CARD\_ERROR**

Card response Error

### **MMC Handle Structure definition**

MMC\_InitTypeDef

MMC\_TypeDef

## 46 HAL NAND Generic Driver

### 46.1 NAND Firmware driver registers structures

#### 46.1.1 NAND\_IDTypeDef

*NAND\_IDTypeDef* is defined in the `stm32f4xx_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*

#### 46.1.2 NAND\_AddressTypeDef

*NAND\_AddressTypeDef* is defined in the `stm32f4xx_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 Plane address
- *uint16\_t* *NAND\_AddressTypeDef::Block*  
NAND memory Block address

#### 46.1.3 NAND\_DeviceConfigTypeDef

*NAND\_DeviceConfigTypeDef* is defined in the `stm32f4xx_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 addressing or words for 16 bits addressing
- *uint32\_t* *NAND\_DeviceConfigTypeDef::SpareAreaSize*  
NAND memory spare area size measured in bytes for 8 bits addressing 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 plane 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 sequence in the NAND device datasheet

#### 46.1.4 NAND\_HandleTypeDef

***NAND\_HandleTypeDef*** is defined in the `stm32f4xx_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 physical characteristic information structure

## 46.2 NAND Firmware driver API description

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

### 46.2.1 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/FSMC 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 `HAL_NAND_Info_TypeDef` 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.

## 46.2.2 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\(\)\*](#)

## 46.2.3 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\_Read\_Status()*
- *HAL\_NAND\_Address\_Inc()*

#### 46.2.4 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()*

#### 46.2.5 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()*

#### 46.2.6 Detailed description of functions

##### HAL\_NAND\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_NAND\_Init (NAND\_HandleTypeDef \* hnd, FMC\_NAND\_PCC\_TimingTypeDef \* ComSpace\_Timing, FMC\_NAND\_PCC\_TimingTypeDef \* AttSpace\_Timing)**

###### Function description

Perform NAND memory Initialization sequence.

###### Parameters

- **hnd**: 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 \* hnd)**

###### Function description

Perform NAND memory De-Initialization sequence.

###### Parameters

- **hnd**: 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 \* h NAND, NAND\_DeviceConfigTypeDef \* pDeviceConfig)**

**Function description**

Configure the device: Enter the physical parameters of the device.

**Parameters**

- **h NAND:** 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 \* h NAND, NAND\_IDTypeDef \* pNAND\_ID)**

**Function description**

Read the NAND memory electronic signature.

**Parameters**

- **h NAND:** 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 \* h NAND)**

**Function description**

NAND MSP Init.

**Parameters**

- **h NAND:** 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 \* h NAND)**

### 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 \* h NAND, NAND\_AddressTypeDef \* pAddress, uint8\_t \* pBuffer, uint32\_t NumPageToRead)

#### Function description

Read Page(s) from NAND memory block (8-bits addressing)

#### Parameters

- **h NAND:** 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 \* h NAND, NAND\_AddressTypeDef \* pAddress, uint8\_t \* pBuffer, uint32\_t NumPageToWrite)

#### Function description

Write Page(s) to NAND memory block (8-bits addressing)

#### Parameters

- **h NAND:** 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 \* h NAND, NAND\_AddressTypeDef \* pAddress, uint8\_t \* pBuffer, uint32\_t NumSpareAreaToRead)

#### Function description

Read Spare area(s) from NAND memory.

#### Parameters

- **h NAND:** 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 \* hnd, NAND\_AddressTypeDef \* pAddress, uint8\_t \* pBuffer, uint32\_t NumSpareAreaToWrite)

### Function description

Write Spare area(s) to NAND memory.

### Parameters

- **hnd**: 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 \* hnd, NAND\_AddressTypeDef \* pAddress, uint16\_t \* pBuffer, uint32\_t NumPageToRead)

### Function description

Read Page(s) from NAND memory block (16-bits addressing)

### Parameters

- **hnd**: 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 \* hnd, NAND\_AddressTypeDef \* pAddress, uint16\_t \* pBuffer, uint32\_t NumPageToWrite)

### Function description

Write Page(s) to NAND memory block (16-bits addressing)

### Parameters

- **hnd**: 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 \* h NAND, NAND\_AddressTypeDef \* pAddress, uint16\_t \* pBuffer, uint32\_t NumSpareAreaToRead)

#### Function description

Read Spare area(s) from NAND memory (16-bits addressing)

#### Parameters

- **h NAND:** 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 \* h NAND, NAND\_AddressTypeDef \* pAddress, uint16\_t \* pBuffer, uint32\_t NumSpareAreaTowrite)

#### Function description

Write Spare area(s) to NAND memory (16-bits addressing)

#### Parameters

- **h NAND:** 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 \* h NAND, NAND\_AddressTypeDef \* pAddress)

#### Function description

NAND memory Block erase.

#### Parameters

- **h NAND:** 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\_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

### 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

## 46.3 NAND Firmware driver defines

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

### 46.3.1 NAND

NAND

#### *NAND Exported Macros*

#### \_\_HAL\_NAND\_RESET\_HANDLE\_STATE

##### Description:

- Reset NAND handle state.

##### Parameters:

- \_\_HANDLE\_\_: specifies the NAND handle.

##### Return value:

- None

## 47 HAL NOR Generic Driver

### 47.1 NOR Firmware driver registers structures

#### 47.1.1 NOR\_IDTypeDef

*NOR\_IDTypeDef* is defined in the stm32f4xx\_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

#### 47.1.2 NOR\_CFITypeDef

*NOR\_CFITypeDef* is defined in the stm32f4xx\_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*

#### 47.1.3 NOR\_HandleTypeDef

*NOR\_HandleTypeDef* is defined in the stm32f4xx\_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

## 47.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

### 47.2.1 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/FSMC 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.

- **MspDelnitCallback** : NOR MspDelnit. 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 MspDelnit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL\_NOR\_Init and @ref HAL\_NOR\_Delnit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDelnit are not null, the @ref HAL\_NOR\_Init and @ref HAL\_NOR\_Delnit keep and use the user MspInit/MspDelnit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDelnit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/Delnit callbacks can be used during the Init/Delnit. In that case first register the MspInit/MspDelnit user callbacks using @ref HAL\_NOR\_RegisterCallback before calling @ref HAL\_NOR\_Delnit 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.

### 47.2.2 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\\_Delnit\(\)](#)
- [HAL\\_NOR\\_MspInit\(\)](#)
- [HAL\\_NOR\\_MspDelnit\(\)](#)
- [HAL\\_NOR\\_MspWait\(\)](#)

### 47.2.3 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\(\)](#)

### 47.2.4 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\(\)](#)

### 47.2.5 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\(\)](#)

## 47.2.6 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

## 47.3 NOR Firmware driver defines

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

### 47.3.1 NOR

NOR

#### *NOR Exported Macros*

#### \_\_HAL\_NOR\_RESET\_HANDLE\_STATE

**Description:**

- Reset NOR handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the NOR handle.

**Return value:**

- None

## 48 HAL PCCARD Generic Driver

### 48.1 PCCARD Firmware driver registers structures

#### 48.1.1 PCCARD\_HandleTypeDef

*PCCARD\_HandleTypeDef* is defined in the `stm32f4xx_hal_pccard.h`

##### Data Fields

- *FMC\_PCCARD\_TypeDef \* Instance*
- *FMC\_PCCARD\_InitTypeDef Init*
- *\_\_IO HAL\_PCCARD\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

##### Field Documentation

- *FMC\_PCCARD\_TypeDef\* PCCARD\_HandleTypeDef::Instance*  
Register base address for PCCARD device
- *FMC\_PCCARD\_InitTypeDef PCCARD\_HandleTypeDef::Init*  
PCCARD device control configuration parameters
- *\_\_IO HAL\_PCCARD\_StateTypeDef PCCARD\_HandleTypeDef::State*  
PCCARD device access state
- *HAL\_LockTypeDef PCCARD\_HandleTypeDef::Lock*  
PCCARD Lock

### 48.2 PCCARD Firmware driver API description

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

#### 48.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control PCCARD/compact flash memories. It uses the FMC/FSMC layer functions to interface with PCCARD devices. This driver is used for:

- PCCARD/Compact Flash memory configuration sequence using the function `HAL_PCCARD_Init()/HAL_CF_Init()` with control and timing parameters for both common and attribute spaces.
- Read PCCARD/Compact Flash memory maker and device IDs using the function `HAL_PCCARD_Read_ID()/HAL_CF_Read_ID()`. The read information is stored in the `CompactFlash_ID` structure declared by the function caller.
- Access PCCARD/Compact Flash memory by read/write operations using the functions `HAL_PCCARD_Read_Sector()/HAL_PCCARD_Write_Sector()` - `HAL_CF_Read_Sector()/HAL_CF_Write_Sector()`, to read/write sector.
- Perform PCCARD/Compact Flash Reset chip operation using the function `HAL_PCCARD_Reset()/HAL_CF_Reset`.
- Perform PCCARD/Compact Flash erase sector operation using the function `HAL_PCCARD_Erase_Sector()/HAL_CF_Erase_Sector`.
- Read the PCCARD/Compact Flash status operation using the function `HAL_PCCARD_ReadStatus()/HAL_CF_ReadStatus()`.
- You can monitor the PCCARD/Compact Flash device HAL state by calling the function `HAL_PCCARD_GetState()/HAL_CF_GetState()`

**Note:** *This driver is a set of generic APIs which handle standard PCCARD/compact flash operations. If a PCCARD/Compact Flash device contains different operations and/or implementations, it should be implemented separately.*

##### Callback registration

The compilation define `USE_HAL_PCCARD_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions `@ref HAL_PCCARD_RegisterCallback()` to register a user callback, it allows to register following callbacks:

- MspInitCallback : PCCARD MspInit.
- MspDeInitCallback : PCCARD 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\_PCCARD\_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : PCCARD MspInit.
- MspDeInitCallback : PCCARD MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL\_PCCARD\_Init and if the state is HAL\_PCCARD\_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\_PCCARD\_Init and @ref HAL\_PCCARD\_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL\_PCCARD\_Init and @ref HAL\_PCCARD\_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\_PCCARD\_RegisterCallback before calling @ref HAL\_PCCARD\_DeInit or @ref HAL\_PCCARD\_Init function. When The compilation define USE\_HAL\_PCCARD\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

#### 48.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

This section contains the following APIs:

- [HAL\\_PCCARD\\_Init\(\)](#)
- [HAL\\_PCCARD\\_DeInit\(\)](#)
- [HAL\\_PCCARD\\_MspInit\(\)](#)
- [HAL\\_PCCARD\\_MspDeInit\(\)](#)

#### 48.2.3 PCCARD Input and Output functions

This section provides functions allowing to use and control the PCCARD memory

This section contains the following APIs:

- [HAL\\_PCCARD\\_Read\\_ID\(\)](#)
- [HAL\\_PCCARD\\_Read\\_Sector\(\)](#)
- [HAL\\_PCCARD\\_Write\\_Sector\(\)](#)
- [HAL\\_PCCARD\\_Erase\\_Sector\(\)](#)
- [HAL\\_PCCARD\\_Reset\(\)](#)
- [HAL\\_PCCARD\\_IRQHandler\(\)](#)
- [HAL\\_PCCARD\\_ITCallback\(\)](#)

#### 48.2.4 PCCARD State functions

This subsection permits to get in run-time the status of the PCCARD controller and the data flow.

This section contains the following APIs:

- [HAL\\_PCCARD\\_GetState\(\)](#)
- [HAL\\_PCCARD\\_GetStatus\(\)](#)
- [HAL\\_PCCARD\\_ReadStatus\(\)](#)

## 48.2.5 Detailed description of functions

### HAL\_PCCARD\_Init

#### Function name

```
HAL_StatusTypeDef HAL_PCCARD_Init (PCCARD_HandleTypeDef * hpccard,
FMC_NAND_PCC_TimingTypeDef * ComSpaceTiming, FMC_NAND_PCC_TimingTypeDef *
AttSpaceTiming, FMC_NAND_PCC_TimingTypeDef * IOSpaceTiming)
```

#### Function description

Perform the PCCARD memory Initialization sequence.

#### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **ComSpaceTiming**: Common space timing structure
- **AttSpaceTiming**: Attribute space timing structure
- **IOSpaceTiming**: IO space timing structure

#### Return values

- **HAL**: status

### HAL\_PCCARD\_DeInit

#### Function name

```
HAL_StatusTypeDef HAL_PCCARD_DeInit (PCCARD_HandleTypeDef * hpccard)
```

#### Function description

Perform the PCCARD memory De-initialization sequence.

#### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

#### Return values

- **HAL**: status

### HAL\_PCCARD\_MspInit

#### Function name

```
void HAL_PCCARD_MspInit (PCCARD_HandleTypeDef * hpccard)
```

#### Function description

PCCARD MSP Init.

#### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

#### Return values

- **None**:

### HAL\_PCCARD\_MspDeInit

#### Function name

```
void HAL_PCCARD_MspDeInit (PCCARD_HandleTypeDef * hpccard)
```

### Function description

PCCARD MSP DeInit.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

### Return values

- **None**:

### HAL\_PCCARD\_Read\_ID

### Function name

HAL\_StatusTypeDef HAL\_PCCARD\_Read\_ID (PCCARD\_HandleTypeDef \* hpccard, uint8\_t CompactFlash\_ID, uint8\_t \* pStatus)

### Function description

Read Compact Flash's ID.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **CompactFlash\_ID**: Compact flash ID structure.
- **pStatus**: pointer to compact flash status

### Return values

- **HAL**: status

### HAL\_PCCARD\_Write\_Sector

### Function name

HAL\_StatusTypeDef HAL\_PCCARD\_Write\_Sector (PCCARD\_HandleTypeDef \* hpccard, uint16\_t \* pBuffer, uint16\_t SectorAddress, uint8\_t \* pStatus)

### Function description

Write sector to PCCARD memory.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **pBuffer**: pointer to source write buffer
- **SectorAddress**: Sector address to write
- **pStatus**: pointer to PCCARD status

### Return values

- **HAL**: status

### HAL\_PCCARD\_Read\_Sector

### Function name

HAL\_StatusTypeDef HAL\_PCCARD\_Read\_Sector (PCCARD\_HandleTypeDef \* hpccard, uint16\_t \* pBuffer, uint16\_t SectorAddress, uint8\_t \* pStatus)

### Function description

Read sector from PCCARD memory.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **pBuffer**: pointer to destination read buffer
- **SectorAddress**: Sector address to read
- **pStatus**: pointer to PCCARD status

### Return values

- **HAL**: status

### HAL\_PCCARD\_Erase\_Sector

### Function name

**HAL\_StatusTypeDef HAL\_PCCARD\_Erase\_Sector (PCCARD\_HandleTypeDef \* hpccard, uint16\_t SectorAddress, uint8\_t \* pStatus)**

### Function description

Erase sector from PCCARD memory.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
- **SectorAddress**: Sector address to erase
- **pStatus**: pointer to PCCARD status

### Return values

- **HAL**: status

### HAL\_PCCARD\_Reset

### Function name

**HAL\_StatusTypeDef HAL\_PCCARD\_Reset (PCCARD\_HandleTypeDef \* hpccard)**

### Function description

Reset the PCCARD memory.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

### Return values

- **HAL**: status

### HAL\_PCCARD\_IRQHandler

### Function name

**void HAL\_PCCARD\_IRQHandler (PCCARD\_HandleTypeDef \* hpccard)**

### Function description

This function handles PCCARD device interrupt request.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

### Return values

- **HAL**: status



### HAL\_PCCARD\_ITCallback

#### Function name

**void HAL\_PCCARD\_ITCallback (PCCARD\_HandleTypeDef \* hpccard)**

#### Function description

PCCARD interrupt feature callback.

#### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

#### Return values

- **None**:

### HAL\_PCCARD\_GetState

#### Function name

**HAL\_PCCARD\_StateTypeDef HAL\_PCCARD\_GetState (PCCARD\_HandleTypeDef \* hpccard)**

#### Function description

return the PCCARD controller state

#### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

#### Return values

- **HAL**: state

### HAL\_PCCARD\_GetStatus

#### Function name

**HAL\_PCCARD\_StatusTypeDef HAL\_PCCARD\_GetStatus (PCCARD\_HandleTypeDef \* hpccard)**

#### Function description

Get the compact flash memory status.

#### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

#### Return values

- **New**: status of the PCCARD operation. This parameter can be:
  - CompactFlash\_TIMEOUT\_ERROR: when the previous operation generate a Timeout error
  - CompactFlash\_READY: when memory is ready for the next operation

### HAL\_PCCARD\_ReadStatus

#### Function name

**HAL\_PCCARD\_StatusTypeDef HAL\_PCCARD\_ReadStatus (PCCARD\_HandleTypeDef \* hpccard)**

#### Function description

Reads the Compact Flash memory status using the Read status command.

### Parameters

- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.

### Return values

- **The**: status of the Compact Flash memory. This parameter can be:
  - CompactFlash\_BUSY: when memory is busy
  - CompactFlash\_READY: when memory is ready for the next operation
  - CompactFlash\_ERROR: when the previous operation generates error

## 48.3 PCCARD Firmware driver defines

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

### 48.3.1 PCCARD

PCCARD

#### *PCCARD Exported Macros*

#### \_\_HAL\_PCCARD\_RESET\_HANDLE\_STATE

##### **Description:**

- Reset PCCARD handle state.

##### **Parameters:**

- \_\_HANDLE\_\_: specifies the PCCARD handle.

##### **Return value:**

- None

## 49 HAL PCD Generic Driver

### 49.1 PCD Firmware driver registers structures

#### 49.1.1 PCD\_HandleTypeDef

*PCD\_HandleTypeDef* is defined in the stm32f4xx\_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[16]*  
IN endpoint parameters
- *PCD\_EPTypedef PCD\_HandleTypeDef::OUT\_ep[16]*  
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

## 49.2 PCD Firmware driver API description

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

### 49.2.1 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_OTG_FS_CLK_ENABLE();`
    - `__HAL_RCC_USB_OTG_HS_CLK_ENABLE();` (For High Speed Mode)
  - 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();`

### 49.2.2 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()***

### 49.2.3 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()***

#### 49.2.4 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()*

#### 49.2.5 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()*

#### 49.2.6 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\_Msplnit

### Function name

**void HAL\_PCD\_Msplnit (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**

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

## 49.3 PCD Firmware driver defines

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

### 49.3.1 PCD

PCD

#### *PCD Exported Macros*

##### \_\_HAL\_PCD\_ENABLE

\_\_HAL\_PCD\_DISABLE  
\_\_HAL\_PCD\_GET\_FLAG  
\_\_HAL\_PCD\_CLEAR\_FLAG  
\_\_HAL\_PCD\_IS\_INVALID\_INTERRUPT  
\_\_HAL\_PCD\_UNGATE\_PHYCLOCK  
\_\_HAL\_PCD\_GATE\_PHYCLOCK  
\_\_HAL\_PCD\_IS\_PHY\_SUSPENDED  
\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_ENABLE\_IT  
\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_DISABLE\_IT  
\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_GET\_FLAG  
\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_CLEAR\_FLAG  
\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_ENABLE\_RISING\_EDGE  
\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_ENABLE\_IT  
\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_DISABLE\_IT  
\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_GET\_FLAG  
\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_CLEAR\_FLAG  
\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_ENABLE\_RISING\_EDGE

***PCD PHY Module***

PCD\_PHY\_ULPI

PCD\_PHY\_EMBEDDED

PCD\_PHY\_UTMI

***PCD Speed***

PCD\_SPEED\_HIGH

PCD\_SPEED\_HIGH\_IN\_FULL

PCD\_SPEED\_FULL

## 50 HAL PCD Extension Driver

### 50.1 PCDEx Firmware driver API description

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

#### 50.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- *HAL\_PCDEx\_SetTxFiFo()*
- *HAL\_PCDEx\_SetRxFiFo()*
- *HAL\_PCDEx\_ActivateLPM()*
- *HAL\_PCDEx\_DeActivateLPM()*
- *HAL\_PCDEx\_LPM\_Callback()*
- *HAL\_PCDEx\_BCD\_Callback()*

#### 50.1.2 Detailed description of functions

##### HAL\_PCDEx\_SetTxFiFo

###### Function name

HAL\_StatusTypeDef HAL\_PCDEx\_SetTxFiFo (PCD\_HandleTypeDef \* hpcd, uint8\_t fifo, uint16\_t size)

###### Function description

Set Tx FIFO.

###### Parameters

- **hpcd**: PCD handle
- **fifo**: The number of Tx fifo
- **size**: Fifo size

###### Return values

- **HAL**: status

##### HAL\_PCDEx\_SetRxFiFo

###### Function name

HAL\_StatusTypeDef HAL\_PCDEx\_SetRxFiFo (PCD\_HandleTypeDef \* hpcd, uint16\_t size)

###### Function description

Set Rx FIFO.

###### Parameters

- **hpcd**: PCD handle
- **size**: Size of Rx fifo

###### 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\_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



## 51 HAL PWR Generic Driver

### 51.1 PWR Firmware driver registers structures

#### 51.1.1 PWR\_PVDTypeDef

*PWR\_PVDTypeDef* is defined in the `stm32f4xx_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`

### 51.2 PWR Firmware driver API description

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

#### 51.2.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) 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()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.

This section contains the following APIs:

- `HAL_PWR_DeInit()`
- `HAL_PWR_EnableBkUpAccess()`
- `HAL_PWR_DisableBkUpAccess()`

#### 51.2.2 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 the `PWR_CR`).
- A 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_PWR_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

##### Wake-up pin configuration

- Wake-up pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There is one Wake-up pin: Wake-up Pin 1 on PA.00.
  - For STM32F446xx there are two Wake-Up pins: Pin1 on PA.00 and Pin2 on PC.13
  - For STM32F410xx/STM32F412xx/STM32F413xx/STM32F423xx there are three Wake-Up pins: Pin1 on PA.00, Pin2 on PC.00 and Pin3 on PC.01

### Low Power modes configuration

The devices feature 3 low-power modes:

- Sleep mode: Cortex-M4 core stopped, peripherals kept running.
- Stop mode: all clocks are stopped, regulator running, regulator in low power mode
- Standby mode: 1.2V domain powered off.

### Sleep mode

- Entry: The Sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI)` functions with
  - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
  - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction

*Note:*

*The Regulator parameter is not used for the STM32F4 family and is kept as parameter just to maintain compatibility with the lower power families (STM32L).*

- Exit: Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

### Stop mode

In Stop mode, all clocks in the 1.2V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption In Stop mode, FLASH can be powered off before entering the Stop mode using the `HAL_PWREx_EnableFlashPowerDown()` function. It can be switched on again by software after exiting the Stop mode using the `HAL_PWREx_DisableFlashPowerDown()` function.

- Entry: The Stop mode is entered using the `HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON)` function with:
  - Main regulator ON.
  - Low Power regulator ON.
- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

### Standby mode

- The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M4 deep sleep mode, with the voltage regulator disabled. The 1.2V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers, backup SRAM and Standby circuitry. The voltage regulator is OFF.
  - Entry:
    - The Standby mode is entered using the `HAL_PWR_EnterSTANDBYMode()` function.
  - Exit:
    - WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wake-up, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

### Auto-wake-up (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wake-up event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wake-up mode).
- RTC auto-wake-up (AWU) from the Stop and Standby 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 Wake-up event, it is necessary to configure the RTC to generate the RTC Wake-up 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\_PVD\_IRQHandler()*
- *HAL\_PWR\_PVDCallback()*
- *HAL\_PWR\_EnableSleepOnExit()*
- *HAL\_PWR\_DisableSleepOnExit()*
- *HAL\_PWR\_EnableSEVOnPend()*
- *HAL\_PWR\_DisableSEVOnPend()*

### 51.2.3 Detailed description of functions

#### **HAL\_PWR\_DeInit**

##### Function name

**void HAL\_PWR\_DeInit (void )**

##### Function description

Deinitializes 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

Enables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

##### Return values

- **None:**

##### Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

#### **HAL\_PWR\_DisableBkUpAccess**

##### Function name

**void HAL\_PWR\_DisableBkUpAccess (void )**

##### Function description

Disables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

##### Return values

- **None:**

##### Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

### HAL\_PWR\_ConfigPVD

#### Function name

**void HAL\_PWR\_ConfigPVD (PWR\_PVDTypeDef \* sConfigPVD)**

#### Function description

Configures the voltage threshold detected by the Power Voltage Detector(PVD).

#### Parameters

- **sConfigPVD:** pointer to an PWR\_PVDTypeDef structure that contains the configuration information for the PVD.

#### Return values

- **None:**

#### Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

### HAL\_PWR\_EnablePVD

#### Function name

**void HAL\_PWR\_EnablePVD (void )**

#### Function description

Enables the Power Voltage Detector(PVD).

#### Return values

- **None:**

### HAL\_PWR\_DisablePVD

#### Function name

**void HAL\_PWR\_DisablePVD (void )**

#### Function description

Disables the Power Voltage Detector(PVD).

#### Return values

- **None:**

### HAL\_PWR\_EnableWakeUpPin

#### Function name

**void HAL\_PWR\_EnableWakeUpPin (uint32\_t WakeUpPinx)**

#### Function description

Enables the Wake-up PINx functionality.

#### Parameters

- **WakeUpPinx:** Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values:
  - PWR\_WAKEUP\_PIN1
  - PWR\_WAKEUP\_PIN2 available only on STM32F410xx/STM32F446xx/STM32F412xx/STM32F413xx/STM32F423xx devices
  - PWR\_WAKEUP\_PIN3 available only on STM32F410xx/STM32F412xx/STM32F413xx/STM32F423xx devices

**Return values**

- **None:**

**HAL\_PWR\_DisableWakeUpPin**

**Function name**

**void HAL\_PWR\_DisableWakeUpPin (uint32\_t WakeUpPinx)**

**Function description**

Disables the Wake-up 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 available only on STM32F410xx/STM32F446xx/STM32F412xx/STM32F413xx/STM32F423xx devices
  - PWR\_WAKEUP\_PIN3 available only on STM32F410xx/STM32F412xx/STM32F413xx/STM32F423xx devices

**Return values**

- **None:**

**HAL\_PWR\_EnterSTOPMode**

**Function name**

**void HAL\_PWR\_EnterSTOPMode (uint32\_t Regulator, uint8\_t STOPEntry)**

**Function description**

Enters Stop mode.

**Parameters**

- **Regulator:** Specifies the regulator state in Stop mode. This parameter can be one of the following values:
  - PWR\_MAINREGULATOR\_ON: Stop mode with regulator ON
  - PWR\_LOWPOWERREGULATOR\_ON: Stop mode with low power regulator ON
- **STOPEntry:** Specifies if Stop mode is 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 mode, all I/O pins keep the same state as in Run mode.
- When exiting Stop mode by issuing an interrupt or a wake-up event, the HSI RC oscillator is selected as system clock.
- When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

**HAL\_PWR\_EnterSLEEPMode**

**Function name**

**void HAL\_PWR\_EnterSLEEPMode (uint32\_t Regulator, uint8\_t SLEEPEntry)**

### Function description

Enters Sleep mode.

### Parameters

- **Regulator:** Specifies the regulator state in SLEEP mode. This parameter can be one of the following values:
  - PWR\_MAINREGULATOR\_ON: SLEEP mode with regulator ON
  - PWR\_LOWPOWERREGULATOR\_ON: SLEEP mode with low power regulator ON
- **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 mode with WFI instruction
  - PWR\_SLEEPENTRY\_WFE: enter SLEEP mode with WFE instruction

### Return values

- **None:**

### Notes

- In Sleep mode, all I/O pins keep the same state as in Run mode.
- In Sleep mode, the systick is stopped to avoid exit from this mode with systick interrupt when used as time base for Timeout
- This parameter is not used for the STM32F4 family and is kept as parameter just to maintain compatibility with the lower power families.

#### HAL\_PWR\_EnterSTANDBYMode

### Function name

**void HAL\_PWR\_EnterSTANDBYMode (void )**

### Function description

Enters Standby mode.

### Return values

- **None:**

### Notes

- In Standby mode, all I/O pins are high impedance except for: Reset pad (still available) RTC\_AF1 pin (PC13) if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out. RTC\_AF2 pin (PI8) if configured for tamper or time-stamp. WKUP pin 1 (PA0) if enabled.

#### HAL\_PWR\_PVD\_IRQHandler

### Function name

**void HAL\_PWR\_PVD\_IRQHandler (void )**

### Function description

This function handles the PWR PVD interrupt request.

### Return values

- **None:**

### Notes

- This API should be called under the PVD\_IRQHandler().

#### HAL\_PWR\_PVDCallback

### Function name

**void HAL\_PWR\_PVDCallback (void )**

**Function description**

PWR PVD interrupt callback.

**Return values**

- **None:**

**HAL\_PWR\_EnableSleepOnExit**
**Function name**

**void HAL\_PWR\_EnableSleepOnExit (void )**

**Function description**

Indicates 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**

Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.

**Return values**

- **None:**

**Notes**

- Clears 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**

Enables CORTEX M4 SEVONPEND bit.

**Return values**

- **None:**

**Notes**

- Sets 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**

Disables CORTEX M4 SEVONPEND bit.

## Return values

- **None:**

## Notes

- Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

## 51.3 PWR Firmware driver defines

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

### 51.3.1 PWR

PWR

*PWR CR Register alias address*

DBP\_BIT\_NUMBER

CR\_DBP\_BB

PVDE\_BIT\_NUMBER

CR\_PVDE\_BB

VOS\_BIT\_NUMBER

CR\_VOS\_BB

*PWR CSR Register alias address*

EWUP\_BIT\_NUMBER

CSR\_EWUP\_BB

*PWR Exported Macro*

\_\_HAL\_PWR\_GET\_FLAG

#### Description:

- Check PWR flag is set or not.

#### Parameters:

- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - PWR\_FLAG\_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
  - PWR\_FLAG\_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  - PWR\_FLAG\_PVDO: PVD Output. This flag is valid only if PVD is enabled by the HAL\_PWR\_EnablePVD() function. The PVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.
  - PWR\_FLAG\_BRR: Backup regulator ready flag. This bit is not reset when the device wakes up from Standby mode or by a system reset or power reset.
  - PWR\_FLAG\_VOSRDY: This flag indicates that the Regulator voltage scaling output selection is ready.

#### Return value:

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).



**\_\_HAL\_PWR\_CLEAR\_FLAG**
**Description:**

- Clear the PWR's pending flags.

**Parameters:**

- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
  - `PWR_FLAG_WU`: Wake Up flag
  - `PWR_FLAG_SB`: StandBy flag

**\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_IT**
**Description:**

- Enable the PVD Exti Line 16.

**Return value:**

- None.

**\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_IT**
**Description:**

- Disable the PVD EXTI Line 16.

**Return value:**

- None.

**\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_EVENT**
**Description:**

- Enable event on PVD Exti Line 16.

**Return value:**

- None.

**\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_EVENT**
**Description:**

- Disable event on PVD Exti Line 16.

**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:**

- PVD EXTI line configuration: set rising & falling edge 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_GET_FLAG`

**Description:**

- checks whether the specified PVD Exti interrupt flag is set or not.

**Return value:**

- EXTI: PVD Line Status.

#### `__HAL_PWR_PVD_EXTI_CLEAR_FLAG`

**Description:**

- Clear the PVD Exti flag.

**Return value:**

- None.

#### `__HAL_PWR_PVD_EXTI_GENERATE_SWIT`

**Description:**

- Generates a Software interrupt on PVD EXTI line.

**Return value:**

- None

**PWR Flag**

`PWR_FLAG_WU`

`PWR_FLAG_SB`

`PWR_FLAG_PVDO`

`PWR_FLAG_BRR`

`PWR_FLAG_VOSRDY`

***PWR Private macros to check input parameters***

`IS_PWR_PVD_LEVEL`

`IS_PWR_PVD_MODE`

IS\_PWR\_REGULATOR

IS\_PWR\_SLEEP\_ENTRY

IS\_PWR\_STOP\_ENTRY

***PWR PVD detection level***

PWR\_PVDLEVEL\_0

PWR\_PVDLEVEL\_1

PWR\_PVDLEVEL\_2

PWR\_PVDLEVEL\_3

PWR\_PVDLEVEL\_4

PWR\_PVDLEVEL\_5

PWR\_PVDLEVEL\_6

PWR\_PVDLEVEL\_7

***PWR PVD EXTI Line***

PWR\_EXTI\_LINE\_PVD

External interrupt line 16 Connected to the PVD EXTI Line

***PWR PVD 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

PVD\_MODE\_EVT

PVD\_RISING\_EDGE

PVD\_FALLING\_EDGE

*PWR Register alias address*

PWR\_OFFSET

PWR\_CR\_OFFSET

PWR\_CSR\_OFFSET

PWR\_CR\_OFFSET\_BB

PWR\_CSR\_OFFSET\_BB

*PWR Regulator state in SLEEP/STOP mode*

PWR\_MAINREGULATOR\_ON

PWR\_LOWPOWERREGULATOR\_ON

*PWR SLEEP mode entry*

PWR\_SLEEPENTRY\_WFI

PWR\_SLEEPENTRY\_WFE

*PWR STOP mode entry*

PWR\_STOPENTRY\_WFI

PWR\_STOPENTRY\_WFE

*PWR WakeUp Pins*

PWR\_WAKEUP\_PIN1

## 52 HAL PWR Extension Driver

### 52.1 PWREx Firmware driver API description

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

#### 52.1.1 Peripheral extended features functions

##### Main and Backup Regulators configuration

- The backup domain includes 4 Kbytes of backup SRAM accessible only from the CPU, and address in 32-bit, 16-bit or 8-bit mode. Its content is retained even in Standby or VBAT mode when the low power backup regulator is enabled. It can be considered as an internal EEPROM when VBAT is always present. You can use the `HAL_PWREx_EnableBkUpReg()` function to enable the low power backup regulator.
- When the backup domain is supplied by VDD (analog switch connected to VDD) the backup SRAM is powered from VDD which replaces the VBAT power supply to save battery life.
- The backup SRAM is not mass erased by a tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested.

*Note:* Refer to the description of Read protection (RDP) in the Flash programming manual.

- The main internal regulator can be configured to have a tradeoff between performance and power consumption when the device does not operate at the maximum frequency. This is done through `__HAL_PWR_MAINREGULATORMODE_CONFIG()` macro which configure VOS bit in PWR\_CR register Refer to the product datasheets for more details.

##### FLASH Power Down configuration

- By setting the FPDS bit in the PWR\_CR register by using the `HAL_PWREx_EnableFlashPowerDown()` function, the Flash memory also enters power down mode when the device enters Stop mode. When the Flash memory is in power down mode, an additional startup delay is incurred when waking up from Stop mode.
- For STM32F42xxx/43xxx/446xx/469xx/479xx Devices, the scale can be modified only when the PLL is OFF and the HSI or HSE clock source is selected as system clock. The new value programmed is active only when the PLL is ON. When the PLL is OFF, the voltage scale 3 is automatically selected. Refer to the datasheets for more details.

##### Over-Drive and Under-Drive configuration

- For STM32F42xxx/43xxx/446xx/469xx/479xx Devices, in Run mode: the main regulator has 2 operating modes available:
  - Normal mode: The CPU and core logic operate at maximum frequency at a given voltage scaling (scale 1, scale 2 or scale 3)
  - Over-drive mode: This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3). This mode is enabled through `HAL_PWREx_EnableOverDrive()` function and disabled by `HAL_PWREx_DisableOverDrive()` function, to enter or exit from Over-drive mode please follow the sequence described in Reference manual.
- For STM32F42xxx/43xxx/446xx/469xx/479xx Devices, in Stop mode: the main regulator or low power regulator supplies a low power voltage to the 1.2V domain, thus preserving the content of registers and internal SRAM. 2 operating modes are available:
  - Normal mode: the 1.2V domain is preserved in nominal leakage mode. This mode is only available when the main regulator or the low power regulator is used in Scale 3 or low voltage mode.
  - Under-drive mode: the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode.

This section contains the following APIs:

- [`HAL\_PWREx\_EnableBkUpReg\(\)`](#)
- [`HAL\_PWREx\_DisableBkUpReg\(\)`](#)
- [`HAL\_PWREx\_EnableFlashPowerDown\(\)`](#)

- *HAL\_PWREx\_DisableFlashPowerDown()*
- *HAL\_PWREx\_GetVoltageRange()*
- *HAL\_PWREx\_ControlVoltageScaling()*
- *HAL\_PWREx\_EnableOverDrive()*
- *HAL\_PWREx\_DisableOverDrive()*
- *HAL\_PWREx\_EnterUnderDriveSTOPMode()*

### 52.1.2 Detailed description of functions

#### **HAL\_PWREx\_EnableFlashPowerDown**

##### Function name

**void HAL\_PWREx\_EnableFlashPowerDown (void )**

##### Function description

Enables the Flash Power Down in Stop mode.

##### Return values

- **None:**

#### **HAL\_PWREx\_DisableFlashPowerDown**

##### Function name

**void HAL\_PWREx\_DisableFlashPowerDown (void )**

##### Function description

Disables the Flash Power Down in Stop mode.

##### Return values

- **None:**

#### **HAL\_PWREx\_EnableBkUpReg**

##### Function name

**HAL\_StatusTypeDef HAL\_PWREx\_EnableBkUpReg (void )**

##### Function description

Enables the Backup Regulator.

##### Return values

- **HAL:** status

#### **HAL\_PWREx\_DisableBkUpReg**

##### Function name

**HAL\_StatusTypeDef HAL\_PWREx\_DisableBkUpReg (void )**

##### Function description

Disables the Backup Regulator.

##### Return values

- **HAL:** status

#### **HAL\_PWREx\_GetVoltageRange**

##### Function name

**uint32\_t HAL\_PWREx\_GetVoltageRange (void )**

### Function description

Return Voltage Scaling Range.

### Return values

- **The:** configured scale for the regulator voltage(VOS bit field). The returned value can be one of the following:
  - ◦ PWR\_REGULATOR\_VOLTAGE\_SCALE1: Regulator voltage output Scale 1 mode
  - ◦ PWR\_REGULATOR\_VOLTAGE\_SCALE2: Regulator voltage output Scale 2 mode
  - ◦ PWR\_REGULATOR\_VOLTAGE\_SCALE3: Regulator voltage output Scale 3 mode

### HAL\_PWREx\_ControlVoltageScaling

#### Function name

**HAL\_StatusTypeDef HAL\_PWREx\_ControlVoltageScaling (uint32\_t VoltageScaling)**

#### Function description

Configures 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: Regulator voltage output range 1 mode, the maximum value of fHCLK is 168 MHz. It can be extended to 180 MHz by activating the over-drive mode.
  - PWR\_REGULATOR\_VOLTAGE\_SCALE2: Regulator voltage output range 2 mode, the maximum value of fHCLK is 144 MHz. It can be extended to, 168 MHz by activating the over-drive mode.
  - PWR\_REGULATOR\_VOLTAGE\_SCALE3: Regulator voltage output range 3 mode, the maximum value of fHCLK is 120 MHz.

#### Return values

- **HAL:** Status

#### Notes

- To update the system clock frequency(SYSCLK): Set the HSI or HSE as system clock frequency using the HAL\_RCC\_ClockConfig(). Call the HAL\_RCC\_OscConfig() to configure the PLL. Call HAL\_PWREx\_ConfigVoltageScaling() API to adjust the voltage scale. Set the new system clock frequency using the HAL\_RCC\_ClockConfig().
- The scale can be modified only when the HSI or HSE clock source is selected as system clock source, otherwise the API returns HAL\_ERROR.
- When the PLL is OFF, the voltage scale 3 is automatically selected and the VOS bits value in the PWR\_CR1 register are not taken in account.
- This API forces the PLL state ON to allow the possibility to configure the voltage scale 1 or 2.
- The new voltage scale is active only when the PLL is ON.

### HAL\_PWREx\_EnableOverDrive

#### Function name

**HAL\_StatusTypeDef HAL\_PWREx\_EnableOverDrive (void )**

#### Function description

Activates the Over-Drive mode.

#### Return values

- **HAL:** status

**Notes**

- This function can be used only for STM32F42xx/STM32F43xx/STM32F446xx/STM32F469xx/STM32F479xx devices. This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3).
- It is recommended to enter or exit Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. During the Over-drive switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.

**HAL\_PWREx\_DisableOverDrive**
**Function name**
**HAL\_StatusTypeDef HAL\_PWREx\_DisableOverDrive (void )**
**Function description**

Deactivates the Over-Drive mode.

**Return values**

- **HAL:** status

**Notes**

- This function can be used only for STM32F42xx/STM32F43xx/STM32F446xx/STM32F469xx/STM32F479xx devices. This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3).
- It is recommended to enter or exit Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. During the Over-drive switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.

**HAL\_PWREx\_EnterUnderDriveSTOPMode**
**Function name**
**HAL\_StatusTypeDef HAL\_PWREx\_EnterUnderDriveSTOPMode (uint32\_t Regulator, uint8\_t STOPEntry)**
**Function description**

Enters in Under-Drive STOP mode.

**Parameters**

- **Regulator:** specifies the regulator state in STOP mode. This parameter can be one of the following values:
  - PWR\_MAINREGULATOR\_UNDERDRIVE\_ON: Main Regulator in under-drive mode and Flash memory in power-down when the device is in Stop under-drive mode
  - PWR\_LOWPOWERREGULATOR\_UNDERDRIVE\_ON: Low Power Regulator in under-drive mode and Flash memory in power-down when the device is in Stop under-drive mode
- **STOPEntry:** specifies if STOP mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR\_SLEEPENTRY\_WFI: enter STOP mode with WFI instruction
  - PWR\_SLEEPENTRY\_WFE: enter STOP mode with WFE instruction

**Return values**

- **None:**



**Notes**

- This mode is only available for STM32F42xxx/STM32F43xxx/STM32F446xx/STM32F469xx/STM32F479xx devices.
- This mode can be selected only when the Under-Drive is already active
- This mode is enabled only with STOP low power mode. In this mode, the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode
- If the Under-drive mode was enabled, it is automatically disabled after exiting Stop mode. When the voltage regulator operates in Under-drive mode, an additional startup delay is induced when waking up from Stop mode.
- In Stop mode, all I/O pins keep the same state as in Run mode.
- When exiting Stop mode by issuing an interrupt or a wake-up event, the HSI RC oscillator is selected as system clock.
- When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

## 52.2 PWREx Firmware driver defines

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

### 52.2.1 PWREx

PWREx

*PWRx CSR Register alias address*

#### BRE\_BIT\_NUMBER

#### CSR\_BRE\_BB

*PWREx Exported Constants*

#### \_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG

**Description:**

- macros configure the main internal regulator output voltage.

**Parameters:**

- `__REGULATOR__`: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheets for more details). This parameter can be one of the following values:
  - `PWR_REGULATOR_VOLTAGE_SCALE1`: Regulator voltage output Scale 1 mode
  - `PWR_REGULATOR_VOLTAGE_SCALE2`: Regulator voltage output Scale 2 mode
  - `PWR_REGULATOR_VOLTAGE_SCALE3`: Regulator voltage output Scale 3 mode

**Return value:**

- None

#### \_\_HAL\_PWR\_OVERDRIVE\_ENABLE

**Notes:**

- These macros can be used only for STM32F42xx/STM3243xx devices.

#### \_\_HAL\_PWR\_OVERDRIVE\_DISABLE

#### \_\_HAL\_PWR\_OVERDRIVESWITCHING\_ENABLE

**Notes:**

- These macros can be used only for STM32F42xx/STM3243xx devices.

#### \_\_HAL\_PWR\_OVERDRIVESWITCHING\_DISABLE

### **\_\_HAL\_PWR\_UNDERDRIVE\_ENABLE**

**Notes:**

- This mode is enabled only with STOP low power mode. In this mode, the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode. If the Under-drive mode was enabled, it is automatically disabled after exiting Stop mode. When the voltage regulator operates in Under-drive mode, an additional startup delay is induced when waking up from Stop mode.

### **\_\_HAL\_PWR\_UNDERDRIVE\_DISABLE**

### **\_\_HAL\_PWR\_GET\_ODRUDR\_FLAG**

**Description:**

- Check PWR flag is set or not.

**Parameters:**

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `PWR_FLAG_ODRDY`: This flag indicates that the Over-drive mode is ready
  - `PWR_FLAG_ODSWRDY`: This flag indicates that the Over-drive mode switching is ready
  - `PWR_FLAG_UDRDY`: This flag indicates that the Under-drive mode is enabled in Stop mode

**Return value:**

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

**Notes:**

- These macros can be used only for STM32F42xx/STM3243xx devices.

### **\_\_HAL\_PWR\_CLEAR\_ODRUDR\_FLAG**

**Notes:**

- These macros can be used only for STM32F42xx/STM3243xx devices.

***PWREx Private macros to check input parameters***

### **IS\_PWR\_REGULATOR\_UNDERDRIVE**

### **IS\_PWR\_VOLTAGE\_SCALING\_RANGE**

### **IS\_PWR\_WAKEUP\_PIN**

***PWREx Over Under Drive Flag***

### **PWR\_FLAG\_ODRDY**

### **PWR\_FLAG\_ODSWRDY**

### **PWR\_FLAG\_UDRDY**

***PWREx Register alias address***

### **FPDS\_BIT\_NUMBER**

### **CR\_FPDS\_BB**

### **ODEN\_BIT\_NUMBER**

### **CR\_ODEN\_BB**

### **ODSWEN\_BIT\_NUMBER**

### **CR\_ODSWEN\_BB**

MRLVDS\_BIT\_NUMBER

CR\_MRLVDS\_BB

LPLVDS\_BIT\_NUMBER

CR\_LPLVDS\_BB

*PWREx Regulator state in UnderDrive mode*

PWR\_MAINREGULATOR\_UNDERDRIVE\_ON

PWR\_LOWPOWERREGULATOR\_UNDERDRIVE\_ON

*PWREx Regulator Voltage Scale*

PWR\_REGULATOR\_VOLTAGE\_SCALE1

PWR\_REGULATOR\_VOLTAGE\_SCALE2

PWR\_REGULATOR\_VOLTAGE\_SCALE3

## 53 HAL QSPI Generic Driver

### 53.1 QSPI Firmware driver registers structures

#### 53.1.1 QSPI\_InitTypeDef

*QSPI\_InitTypeDef* is defined in the `stm32f4xx_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*

#### 53.1.2 QSPI\_HandleTypeDef

*QSPI\_HandleTypeDef* is defined in the `stm32f4xx_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`

### 53.1.3

#### QSPI\_CommandTypeDef

`QSPI_CommandTypeDef` is defined in the `stm32f4xx_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`

### 53.1.4

#### QSPI\_AutoPollingTypeDef

`QSPI_AutoPollingTypeDef` is defined in the `stm32f4xx_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*

**53.1.5**
**QSPI\_MemoryMappedTypeDef**

*QSPI\_MemoryMappedTypeDef* is defined in the `stm32f4xx_hal_qspi.h`

**Data Fields**

- *uint32\_t TimeOutPeriod*
- *uint32\_t TimeOutActivation*

**Field Documentation**

- *uint32\_t QSPI\_MemoryMappedTypeDef::TimeOutPeriod*
- *uint32\_t QSPI\_MemoryMappedTypeDef::TimeOutActivation*

**53.2**
**QSPI Firmware driver API description**

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

**53.2.1**
**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 period 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 period 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

### 53.2.2 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\(\)\*](#)

### 53.2.3 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()*

### 53.2.4 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()*

### 53.2.5 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.

## 53.3 QSPI Firmware driver defines

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

### 53.3.1 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 half of system 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**

## 54 HAL RCC Generic Driver

### 54.1 RCC Firmware driver registers structures

#### 54.1.1 RCC\_OscInitTypeDef

*RCC\_OscInitTypeDef* is defined in the stm32f4xx\_hal\_rcc.h

##### Data Fields

- *uint32\_t OscillatorType*
- *uint32\_t HSEState*
- *uint32\_t LSEState*
- *uint32\_t HSIState*
- *uint32\_t HSI CalibrationValue*
- *uint32\_t LSIState*
- *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::HSI CalibrationValue*  
The HSI calibration trimming value (default is RCC\_HSI CALIBRATION\_DEFAULT). This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- *uint32\_t RCC\_OscInitTypeDef::LSIState*  
The new state of the LSI. This parameter can be a value of [RCC\\_LSI\\_Config](#)
- *RCC\_PLLInitTypeDef RCC\_OscInitTypeDef::PLL*  
PLL structure parameters

#### 54.1.2 RCC\_ClkInitTypeDef

*RCC\_ClkInitTypeDef* is defined in the stm32f4xx\_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 (SYSCLKS) used as system clock. 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](#)

## 54.2 RCC Firmware driver API description

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

### 54.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 16MHz) 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 (AHB) and Low speed (APB) 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 input floating state, 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 (I2S, RTC, ADC, USB OTG FS/SDIO/RNG)

### 54.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Implemented Workaround:

- For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `__HAL_RCC_PPP_CLK_ENABLE()` macro.

### 54.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/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

1. HSI (high-speed internal), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. LSI (low-speed internal), 32 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 26 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
4. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.

5. PLL (clocked by HSI or HSE), featuring two different output clocks:
  - The first output is used to generate the high speed system clock (up to 168 MHz)
  - The second output is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDIO (<= 48 MHz).
6. CSS (Clock security system), once enable using the macro `__HAL_RCC_CSS_ENABLE()` and if a HSE clock failure occurs(HSE used directly or through PLL as System clock source), the System clocks 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.
7. MCO1 (microcontroller clock output), used to output HSI, LSE, HSE or PLL clock (through a configurable prescaler) on PA8 pin.
8. MCO2 (microcontroller clock output), used to output HSE, PLL, SYSCLK or PLLI2S clock (through a configurable prescaler) on PC9 pin.

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and 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.
2. For the STM32F405xx/07xx and STM32F415xx/17xx devices, the maximum frequency of the SYSCLK and HCLK is 168 MHz, PCLK2 84 MHz and PCLK1 42 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).
3. For the STM32F42xxx, STM32F43xxx, STM32F446xx, STM32F469xx and STM32F479xx devices, the maximum frequency of the SYSCLK and HCLK is 180 MHz, PCLK2 90 MHz and PCLK1 45 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).
4. For the STM32F401xx, the maximum frequency of the SYSCLK and HCLK is 84 MHz, PCLK2 84 MHz and PCLK1 42 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).
5. For the STM32F41xxx, the maximum frequency of the SYSCLK and HCLK is 100 MHz, PCLK2 100 MHz and PCLK1 50 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).

This section contains the following APIs:

- [\*HAL\\_RCC\\_DeInit\(\)\*](#)
- [\*HAL\\_RCC\\_OscConfig\(\)\*](#)
- [\*HAL\\_RCC\\_ClockConfig\(\)\*](#)

#### 54.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- [\*HAL\\_RCC\\_MCOConfig\(\)\*](#)
- [\*HAL\\_RCC\\_EnableCSS\(\)\*](#)
- [\*HAL\\_RCC\\_DisableCSS\(\)\*](#)
- [\*HAL\\_RCC\\_GetSysClockFreq\(\)\*](#)
- [\*HAL\\_RCC\\_GetHCLKFreq\(\)\*](#)
- [\*HAL\\_RCC\\_GetPCLK1Freq\(\)\*](#)
- [\*HAL\\_RCC\\_GetPCLK2Freq\(\)\*](#)
- [\*HAL\\_RCC\\_GetOscConfig\(\)\*](#)
- [\*HAL\\_RCC\\_GetClockConfig\(\)\*](#)
- [\*HAL\\_RCC\\_NMI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_RCC\\_CSSCallback\(\)\*](#)

## 54.2.5 Detailed description of functions

### HAL\_RCC\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_RCC\_DeInit (void )**

#### Function description

Resets 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 source HSE and PLL OFF AHB, APB1 and APB2 prescaler set to 1.CSS, MCO1 and MCO2 OFF All interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocks LSI, LSE and RTC clocks

### HAL\_RCC\_OscConfig

#### Function name

**HAL\_StatusTypeDef HAL\_RCC\_OscConfig (RCC\_OscInitTypeDef \* RCC\_OscInitStruct)**

#### Function description

Initializes 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 API. 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 API. 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

Initializes 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 depend on device selected

#### 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 (enabled by hardware) as system clock source after startup from Reset, wake-up 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).
- 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 will be ready.
- 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

Selects the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).

### Parameters

- **RCC\_MCOx**: specifies the output direction for the clock source. This parameter can be one of the following values:
  - RCC\_MCO1: Clock source to output on MCO1 pin(PA8).
  - RCC\_MCO2: Clock source to output on MCO2 pin(PC9).
- **RCC\_MCOSource**: specifies the clock source to output. This parameter can be one of the following values:
  - RCC\_MCO1SOURCE\_HSI: HSI clock selected as MCO1 source
  - RCC\_MCO1SOURCE\_LSE: LSE clock selected as MCO1 source
  - RCC\_MCO1SOURCE\_HSE: HSE clock selected as MCO1 source
  - RCC\_MCO1SOURCE\_PLLCLK: main PLL clock selected as MCO1 source
  - RCC\_MCO2SOURCE\_SYCLK: System clock (SYSCLK) selected as MCO2 source
  - RCC\_MCO2SOURCE\_PLLI2SCLK: PLLI2S clock selected as MCO2 source, available for all STM32F4 devices except STM32F410xx
  - RCC\_MCO2SOURCE\_I2SCLK: I2SCLK clock selected as MCO2 source, available only for STM32F410Rx devices
  - RCC\_MCO2SOURCE\_HSE: HSE clock selected as MCO2 source
  - RCC\_MCO2SOURCE\_PLLCLK: main PLL clock selected as MCO2 source
- **RCC\_MCODiv**: specifies the MCOx prescaler. This parameter can be one of the following values:
  - RCC\_MCODIV\_1: no division applied to MCOx clock
  - RCC\_MCODIV\_2: division by 2 applied to MCOx clock
  - RCC\_MCODIV\_3: division by 3 applied to MCOx clock
  - RCC\_MCODIV\_4: division by 4 applied to MCOx clock
  - RCC\_MCODIV\_5: division by 5 applied to MCOx clock

### Return values

- **None**:

### Notes

- PA8/PC9 should be configured in alternate function mode.
- For STM32F410Rx devices to output I2SCLK clock on MCO2 you should have at last one of the SPI clocks enabled (SPI1, SPI2 or SPI5).

### HAL\_RCC\_EnableCSS

#### Function name

**void HAL\_RCC\_EnableCSS (void )**

#### Function description

Enables 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.

### HAL\_RCC\_DisableCSS

#### Function name

**void HAL\_RCC\_DisableCSS (void )**

#### Function description

Disables the Clock Security System.

#### Return values

- **None:**

### HAL\_RCC\_GetSysClockFreq

#### Function name

**uint32\_t HAL\_RCC\_GetSysClockFreq (void )**

#### Function description

Returns 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(\*\*) or HSI\_VALUE(\*) multiplied/divided by the PLL factors.
- (\*) HSI\_VALUE is a constant defined in stm32f4xx\_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 stm32f4xx\_hal\_conf.h file (default value 25 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

Returns the HCLK frequency.

#### Return values

- **HCLK:** frequency

#### 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 and updated within this function

### HAL\_RCC\_GetPCLK1Freq

#### Function name

**uint32\_t HAL\_RCC\_GetPCLK1Freq (void )**

#### Function description

Returns the PCLK1 frequency.

#### Return values

- **PCLK1:** frequency

#### 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

Returns the PCLK2 frequency.

#### Return values

- **PCLK2:** frequency

#### 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

Configures 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**

Configures 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**

This function handles the RCC CSS 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:**

## 54.3 RCC Firmware driver defines

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

### 54.3.1 RCC

RCC

*AHB1 Peripheral Clock Enable Disable*
**\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE**
**\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE**
**\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE**
**\_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE**

\_\_HAL\_RCC\_DMA1\_CLK\_ENABLE

\_\_HAL\_RCC\_DMA2\_CLK\_ENABLE

\_\_HAL\_RCC\_GPIOA\_CLK\_DISABLE

\_\_HAL\_RCC\_GPIOB\_CLK\_DISABLE

\_\_HAL\_RCC\_GPIOC\_CLK\_DISABLE

\_\_HAL\_RCC\_GPIOH\_CLK\_DISABLE

\_\_HAL\_RCC\_DMA1\_CLK\_DISABLE

\_\_HAL\_RCC\_DMA2\_CLK\_DISABLE

***AHB1 Force Release Reset***

\_\_HAL\_RCC\_AHB1\_FORCE\_RESET

\_\_HAL\_RCC\_GPIOA\_FORCE\_RESET

\_\_HAL\_RCC\_GPIOB\_FORCE\_RESET

\_\_HAL\_RCC\_GPIOC\_FORCE\_RESET

\_\_HAL\_RCC\_GPIOH\_FORCE\_RESET

\_\_HAL\_RCC\_DMA1\_FORCE\_RESET

\_\_HAL\_RCC\_DMA2\_FORCE\_RESET

\_\_HAL\_RCC\_AHB1\_RELEASE\_RESET

\_\_HAL\_RCC\_GPIOA\_RELEASE\_RESET

\_\_HAL\_RCC\_GPIOB\_RELEASE\_RESET

\_\_HAL\_RCC\_GPIOC\_RELEASE\_RESET

\_\_HAL\_RCC\_GPIOH\_RELEASE\_RESET

\_\_HAL\_RCC\_DMA1\_RELEASE\_RESET

\_\_HAL\_RCC\_DMA2\_RELEASE\_RESET

***AHB1 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_GPIOA\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_GPIOB\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_GPIOC\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_GPIOH\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_DMA1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_DMA2\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_GPIOA\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_GPIOB\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_GPIOC\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_GPIOH\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_DMA1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_DMA2\_CLK\_SLEEP\_DISABLE

***AHB1 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_GPIOH\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_DMA1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_DMA2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_GPIOH\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_DMA1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_DMA2\_IS\_CLK\_DISABLED

***AHB Clock Source***

RCC\_SYSCLK\_DIV1

RCC\_SYSCLK\_DIV2

RCC\_SYSCLK\_DIV4

RCC\_SYSCLK\_DIV8

RCC\_SYSCLK\_DIV16

RCC\_SYSCLK\_DIV64

RCC\_SYSCLK\_DIV128

RCC\_SYSCLK\_DIV256

RCC\_SYSCLK\_DIV512

***APB1/APB2 Clock Source***

RCC\_HCLK\_DIV1

RCC\_HCLK\_DIV2

RCC\_HCLK\_DIV4

RCC\_HCLK\_DIV8

RCC\_HCLK\_DIV16

***APB1 Peripheral Clock Enable Disable***

\_\_HAL\_RCC\_TIM5\_CLK\_ENABLE

\_\_HAL\_RCC\_WWDG\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI2\_CLK\_ENABLE

\_\_HAL\_RCC\_USART2\_CLK\_ENABLE

\_\_HAL\_RCC\_I2C1\_CLK\_ENABLE

\_\_HAL\_RCC\_I2C2\_CLK\_ENABLE

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM5\_CLK\_DISABLE

\_\_HAL\_RCC\_WWDG\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI2\_CLK\_DISABLE

\_\_HAL\_RCC\_USART2\_CLK\_DISABLE

\_\_HAL\_RCC\_I2C1\_CLK\_DISABLE

\_\_HAL\_RCC\_I2C2\_CLK\_DISABLE

\_\_HAL\_RCC\_PWR\_CLK\_DISABLE

***APB1 Force Release Reset***

\_\_HAL\_RCC\_APB1\_FORCE\_RESET

\_\_HAL\_RCC\_TIM5\_FORCE\_RESET

\_\_HAL\_RCC\_WWDG\_FORCE\_RESET

\_\_HAL\_RCC\_SPI2\_FORCE\_RESET

\_\_HAL\_RCC\_USART2\_FORCE\_RESET

\_\_HAL\_RCC\_I2C1\_FORCE\_RESET

\_\_HAL\_RCC\_I2C2\_FORCE\_RESET

\_\_HAL\_RCC\_PWR\_FORCE\_RESET

\_\_HAL\_RCC\_APB1\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM5\_RELEASE\_RESET

\_\_HAL\_RCC\_WWDG\_RELEASE\_RESET

\_\_HAL\_RCC\_SPI2\_RELEASE\_RESET

\_\_HAL\_RCC\_USART2\_RELEASE\_RESET

\_\_HAL\_RCC\_I2C1\_RELEASE\_RESET

\_\_HAL\_RCC\_I2C2\_RELEASE\_RESET

\_\_HAL\_RCC\_PWR\_RELEASE\_RESET

***APB1 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_TIM5\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_WWDG\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SPI2\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USART2\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_I2C1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_I2C2\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_PWR\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM5\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_WWDG\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_SPI2\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_USART2\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_I2C1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_I2C2\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_PWR\_CLK\_SLEEP\_DISABLE

***APB1 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_TIM5\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_WWDG\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SPI2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_USART2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_I2C1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_I2C2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_PWR\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM5\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_WWDG\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SPI2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_USART2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_I2C1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_I2C2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_PWR\_IS\_CLK\_DISABLED

***APB2 Peripheral Clock Enable Disable***

\_\_HAL\_RCC\_TIM1\_CLK\_ENABLE

\_\_HAL\_RCC\_USART1\_CLK\_ENABLE

\_\_HAL\_RCC\_USART6\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC1\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI1\_CLK\_ENABLE

\_\_HAL\_RCC\_SYSCFG\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM9\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM11\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM1\_CLK\_DISABLE

\_\_HAL\_RCC\_USART1\_CLK\_DISABLE

\_\_HAL\_RCC\_USART6\_CLK\_DISABLE

\_\_HAL\_RCC\_ADC1\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI1\_CLK\_DISABLE

\_\_HAL\_RCC\_SYSCFG\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM9\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM11\_CLK\_DISABLE

***APB2 Force Release Reset***

\_\_HAL\_RCC\_APB2\_FORCE\_RESET

\_\_HAL\_RCC\_TIM1\_FORCE\_RESET

\_\_HAL\_RCC\_USART1\_FORCE\_RESET

\_\_HAL\_RCC\_USART6\_FORCE\_RESET

\_\_HAL\_RCC\_ADC\_FORCE\_RESET

\_\_HAL\_RCC\_SPI1\_FORCE\_RESET

\_\_HAL\_RCC\_SYSCFG\_FORCE\_RESET

\_\_HAL\_RCC\_TIM9\_FORCE\_RESET

\_\_HAL\_RCC\_TIM11\_FORCE\_RESET

\_\_HAL\_RCC\_APB2\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM1\_RELEASE\_RESET

\_\_HAL\_RCC\_USART1\_RELEASE\_RESET

\_\_HAL\_RCC\_USART6\_RELEASE\_RESET

\_\_HAL\_RCC\_ADC\_RELEASE\_RESET

\_\_HAL\_RCC\_SPI1\_RELEASE\_RESET

\_\_HAL\_RCC\_SYSCFG\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM9\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM11\_RELEASE\_RESET

***APB2 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_TIM1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USART1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USART6\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ADC1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SPI1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SYSCFG\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM9\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM11\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_USART1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_USART6\_CLK\_SLEEP\_DISABLE



\_\_HAL\_RCC\_ADC1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_SPI1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_SYSCFG\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_TIM9\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_TIM11\_CLK\_SLEEP\_DISABLE

***APB2 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_TIM1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_USART1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_USART6\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_ADC1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SPI1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SYSCFG\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM9\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM11\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_USART1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_USART6\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_ADC1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SPI1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SYSCFG\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM9\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM11\_IS\_CLK\_DISABLED

***RCC BitAddress AliasRegion***

RCC\_OFFSET

RCC\_CR\_OFFSET

RCC\_HSION\_BIT\_NUMBER

RCC\_CR\_HSION\_BB

RCC\_CSSON\_BIT\_NUMBER

RCC\_CR\_CSSON\_BB

RCC\_PLLON\_BIT\_NUMBER

RCC\_CR\_PLLON\_BB

RCC\_BDCR\_OFFSET

RCC\_RTCEN\_BIT\_NUMBER

RCC\_BDCR\_RTCEN\_BB

RCC\_BDRST\_BIT\_NUMBER

RCC\_BDCR\_BDRST\_BB

RCC\_CSR\_OFFSET

RCC\_LSION\_BIT\_NUMBER

RCC\_CSR\_LSION\_BB

RCC\_CR\_BYTE2\_ADDRESS

RCC\_CIR\_BYTE1\_ADDRESS

RCC\_CIR\_BYTE2\_ADDRESS

RCC\_BDCR\_BYTE0\_ADDRESS

RCC\_DBP\_TIMEOUT\_VALUE

RCC\_LSE\_TIMEOUT\_VALUE

HSE\_TIMEOUT\_VALUE

HSI\_TIMEOUT\_VALUE

LSI\_TIMEOUT\_VALUE

CLOCKSWITCH\_TIMEOUT\_VALUE

### *Flags*

RCC\_FLAG\_HSIRDY

RCC\_FLAG\_HSERDY

RCC\_FLAG\_PLLRDY

RCC\_FLAG\_PLLI2SRDY

RCC\_FLAG\_LSERDY

RCC\_FLAG\_LSIRDY

RCC\_FLAG BORRST

RCC\_FLAG\_PINRST

RCC\_FLAG\_PORRST

RCC\_FLAG\_SFTRST

RCC\_FLAG\_IWDGRST

RCC\_FLAG\_WWDGRST

RCC\_FLAG\_LPWRST

### ***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\_PLLI2SRDY: PLLI2S ready interrupt.

**\_\_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\_PLLI2SRDY: PLLI2S ready interrupt.

**\_\_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\_PLLI2SRDY: PLLI2S ready interrupt.
  - RCC\_IT\_CSS: Clock Security System interrupt

## `__HAL_RCC_GET_IT`

### Description:

- Check the RCC's 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_PLLI2SRDY`: PLLI2S ready interrupt.
  - `RCC_IT_CSS`: Clock Security System interrupt

### Return value:

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

## `__HAL_RCC_CLEAR_RESET_FLAGS`

## `RCC_FLAG_MASK`

### Description:

- Check 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_PLLI2SRDY`: PLLI2S clock ready.
  - `RCC_FLAG_LSERDY`: LSE oscillator clock ready.
  - `RCC_FLAG_LSIRDY`: LSI oscillator clock ready.
  - `RCC_FLAG_BORRST`: POR/PDR or BOR reset.
  - `RCC_FLAG_PINRST`: Pin reset.
  - `RCC_FLAG_PORRST`: POR/PDR 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).

## `__HAL_RCC_GET_FLAG`

### *Get Clock source*

## **\_\_HAL\_RCC\_SYSCLK\_CONFIG**

### **Description:**

- Macro to configure the system clock source.

### **Parameters:**

- **\_\_RCC\_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.
  - **RCC\_SYSCLKSOURCE\_PLLRCLK**: PLLR output is used as system clock source. This parameter is available only for STM32F446xx devices.

## **\_\_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.
  - **RCC\_SYSCLKSOURCE\_STATUS\_PLLRCLK**: PLLR used as system clock. This parameter is available only for STM32F446xx devices.

## **\_\_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\_HSI**: HSI oscillator is used as PLL clock source.
  - **RCC\_PLLSOURCE\_HSE**: HSE oscillator is used as PLL clock source.

### **HSE Config**

#### **RCC\_HSE\_OFF**

#### **RCC\_HSE\_ON**

#### **RCC\_HSE\_BYPASS**

### **HSE Configuration**

## **\_\_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.

### **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.

### **HSI Config**

## **RCC\_HSI\_OFF**

## **RCC\_HSI\_ON**

## **RCC\_HSCALIBRATION\_DEFAULT**

### **HSI Configuration**

## **\_\_HAL\_RCC\_HSI\_ENABLE**

### **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, wake-up 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 oscillator (HSI) calibration value.

### **Parameters:**

- `__HSICalibrationValue__`: specifies the calibration trimming value. (default is `RCC_HSCALIBRATION_DEFAULT`). This parameter must be a number between 0 and 0x1F.

### **Notes:**

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

### **RTC Clock Configuration**

## **\_\_HAL\_RCC\_RTC\_ENABLE**

**Notes:**

- These macros must be used only after the RTC clock source was selected.

## **\_\_HAL\_RCC\_RTC\_DISABLE**

## **\_\_HAL\_RCC\_RTC\_CLKPRESCALER**

**Description:**

- Macros to configure the RTC clock (RTCCLK).

**Parameters:**

- `__RTCCLKSource__`: specifies the RTC clock source. This parameter can be one of the following values:
  - `RCC_RTCCLKSOURCE_NO_CLK` : 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_DIVX` HSE divided by X selected as RTC clock (X can be retrieved thanks to `__HAL_RCC_GET_RTC_HSE_PRESCALER()`)

**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 can't be changed unless the Backup domain is reset using `__HAL_RCC_BackupReset_RELEASE()` 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 wake-up 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\_RTC\_CONFIG**

## **\_\_HAL\_RCC\_GET\_RTC\_SOURCE**

**Description:**

- Macro to get the RTC clock source.

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_RTCCLKSOURCE_NO_CLK` 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_DIVX` HSE divided by X selected as RTC clock (X can be retrieved thanks to `__HAL_RCC_GET_RTC_HSE_PRESCALER()`)

## **\_\_HAL\_RCC\_GET\_RTC\_HSE\_PRESCALER**

**Description:**

- Get the RTC and HSE clock divider (RTCPRE).

**Return value:**

- Returned: value can be one of the following values:
  - `RCC_RTCCLKSOURCE_HSE_DIVX` HSE divided by X selected as RTC clock (X can be retrieved thanks to `__HAL_RCC_GET_RTC_HSE_PRESCALER()`)

## **\_\_HAL\_RCC\_BACKUPRESET\_FORCE**

**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`

*Interrupts*

`RCC_IT_LSIRDY`

`RCC_IT_LSERDY`

`RCC_IT_HSIRDY`

`RCC_IT_HSERDY`

`RCC_IT_PLLRDY`

`RCC_IT_PLLI2SRDY`

`RCC_IT_CSS`

*RCC Private macros to check input parameters*

`IS_RCC_OSCILLATORTYPE`

`IS_RCC_HSE`

`IS_RCC_LSE`

`IS_RCC_HSI`

`IS_RCC_LSI`

`IS_RCC_PLL`

`IS_RCC_PLLSOURCE`

`IS_RCC_SYSCLKSOURCE`

`IS_RCC_RTCCLKSOURCE`

`IS_RCC_PLLM_VALUE`

`IS_RCC_PLLP_VALUE`

`IS_RCC_PLLQ_VALUE`

`IS_RCC_HCLK`

`IS_RCC_CLOCKTYPE`

`IS_RCC_PCLK`

`IS_RCC_MCO`

`IS_RCC_MCO1SOURCE`

`IS_RCC_MCODIV`

`IS_RCC_CALIBRATION_VALUE`

*LSE Config*



RCC\_LSE\_OFF

RCC\_LSE\_ON

RCC\_LSE\_BYPASS

***LSE Configuration***

\_\_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.

**Notes:**

- Transition 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.

***LSI Config***

RCC\_LSI\_OFF

RCC\_LSI\_ON

***LSI Configuration***

\_\_HAL\_RCC\_LSI\_ENABLE

**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

***MCO1 Clock Source***

RCC\_MCO1SOURCE\_HSI

RCC\_MCO1SOURCE\_LSE

RCC\_MCO1SOURCE\_HSE

RCC\_MCO1SOURCE\_PLLCLK

***MCO2 Clock Source***

RCC\_MCO2SOURCE\_SYSCLK

RCC\_MCO2SOURCE\_PLLI2SCLK

RCC\_MCO2SOURCE\_HSE

RCC\_MCO2SOURCE\_PLLCLK

*MCOx Clock Prescaler*

RCC\_MCODIV\_1

RCC\_MCODIV\_2

RCC\_MCODIV\_3

RCC\_MCODIV\_4

RCC\_MCODIV\_5

*MCO Index*

RCC\_MCO1

RCC\_MCO2

*Oscillator Type*

RCC\_OSCILLATORTYPE\_NONE

RCC\_OSCILLATORTYPE\_HSE

RCC\_OSCILLATORTYPE\_HSI

RCC\_OSCILLATORTYPE\_LSE

RCC\_OSCILLATORTYPE\_LSI

*PLL Clock Divider*

RCC\_PLLP\_DIV2

RCC\_PLLP\_DIV4

RCC\_PLLP\_DIV6

RCC\_PLLP\_DIV8

*PLL Clock Source*

RCC\_PLLSOURCE\_HSI

RCC\_PLLSOURCE\_HSE

*PLL Config*

RCC\_PLL\_NONE

RCC\_PLL\_OFF

RCC\_PLL\_ON

*PLL Configuration*

**\_\_HAL\_RCC\_PLL\_ENABLE**
**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\_HSI**: HSI oscillator clock selected as PLL clock entry
  - **RCC\_PLLSOURCE\_HSE**: HSE oscillator clock selected as PLL clock entry

**Notes:**

- This function must be used only when the main PLL is disabled.

**\_\_HAL\_RCC\_PLL\_PLLM\_CONFIG**
**Description:**

- Macro to configure the PLL multiplication factor.

**Parameters:**

- **\_\_PLLM\_\_**: specifies the division factor for PLL VCO input clock. This parameter must be a number between **Min\_Data = 2** and **Max\_Data = 63**.

**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 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

***RTC Clock Source***
**RCC\_RTCCLKSOURCE\_NO\_CLK**
**RCC\_RTCCLKSOURCE\_LSE**
**RCC\_RTCCLKSOURCE\_LSI**
**RCC\_RTCCLKSOURCE\_HSE\_DIVX**
**RCC\_RTCCLKSOURCE\_HSE\_DIV2**
**RCC\_RTCCLKSOURCE\_HSE\_DIV3**
**RCC\_RTCCLKSOURCE\_HSE\_DIV4**
**RCC\_RTCCLKSOURCE\_HSE\_DIV5**
**RCC\_RTCCLKSOURCE\_HSE\_DIV6**
**RCC\_RTCCLKSOURCE\_HSE\_DIV7**
**RCC\_RTCCLKSOURCE\_HSE\_DIV8**

RCC\_RTCCLKSOURCE\_HSE\_DIV9

RCC\_RTCCLKSOURCE\_HSE\_DIV10

RCC\_RTCCLKSOURCE\_HSE\_DIV11

RCC\_RTCCLKSOURCE\_HSE\_DIV12

RCC\_RTCCLKSOURCE\_HSE\_DIV13

RCC\_RTCCLKSOURCE\_HSE\_DIV14

RCC\_RTCCLKSOURCE\_HSE\_DIV15

RCC\_RTCCLKSOURCE\_HSE\_DIV16

RCC\_RTCCLKSOURCE\_HSE\_DIV17

RCC\_RTCCLKSOURCE\_HSE\_DIV18

RCC\_RTCCLKSOURCE\_HSE\_DIV19

RCC\_RTCCLKSOURCE\_HSE\_DIV20

RCC\_RTCCLKSOURCE\_HSE\_DIV21

RCC\_RTCCLKSOURCE\_HSE\_DIV22

RCC\_RTCCLKSOURCE\_HSE\_DIV23

RCC\_RTCCLKSOURCE\_HSE\_DIV24

RCC\_RTCCLKSOURCE\_HSE\_DIV25

RCC\_RTCCLKSOURCE\_HSE\_DIV26

RCC\_RTCCLKSOURCE\_HSE\_DIV27

RCC\_RTCCLKSOURCE\_HSE\_DIV28

RCC\_RTCCLKSOURCE\_HSE\_DIV29

RCC\_RTCCLKSOURCE\_HSE\_DIV30

RCC\_RTCCLKSOURCE\_HSE\_DIV31

***System Clock Source***

RCC\_SYSCLKSOURCE\_HSI

RCC\_SYSCLKSOURCE\_HSE

RCC\_SYSCLKSOURCE\_PLLCLK

RCC\_SYSCLKSOURCE\_PLLRCLK

***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

RCC\_SYSCLKSOURCE\_STATUS\_PLLRCLK

PLLR used as system clock

***System Clock Type***

RCC\_CLOCKTYPE\_SYSCLK

RCC\_CLOCKTYPE\_HCLK

RCC\_CLOCKTYPE\_PCLK1

RCC\_CLOCKTYPE\_PCLK2

## 55 HAL RCC Extension Driver

### 55.1 RCCEX Firmware driver registers structures

#### 55.1.1 RCC\_PLLInitTypeDef

*RCC\_PLLInitTypeDef* is defined in the `stm32f4xx_hal_rcc_ex.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 number between `Min_Data = 0` and `Max_Data = 63`
- *uint32\_t RCC\_PLLInitTypeDef::PLLN*  
PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between `Min_Data = 50` and `Max_Data = 432` except for STM32F411xE devices where the `Min_Data = 192`
- *uint32\_t RCC\_PLLInitTypeDef::PLLP*  
PLLP: Division factor for main system clock (SYSCLK). This parameter must be a value of [RCC\\_PLLP\\_Clock\\_Divider](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLQ*  
PLLQ: Division factor for OTG FS, SDIO and RNG clocks. This parameter must be a number between `Min_Data = 2` and `Max_Data = 15`
- *uint32\_t RCC\_PLLInitTypeDef::PLLR*  
PLLR: PLL division factor for I2S, SAI, SYSTEM, SPDIFRX clocks. This parameter is only available in STM32F410xx/STM32F446xx/STM32F469xx/STM32F479xx and STM32F412Zx/STM32F412Vx/STM32F412Rx/STM32F412Cx/STM32F413xx/STM32F423xx devices. This parameter must be a number between `Min_Data = 2` and `Max_Data = 7`

#### 55.1.2 RCC\_PLLI2SInitTypeDef

*RCC\_PLLI2SInitTypeDef* is defined in the `stm32f4xx_hal_rcc_ex.h`

##### Data Fields

- *uint32\_t PLLI2SN*
- *uint32\_t PLLI2SR*
- *uint32\_t PLLI2SQ*

##### Field Documentation

- *uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SN*  
Specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between `Min_Data = 50` and `Max_Data = 432`. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI

- ***uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SR***  
Specifies the division factor for I2S clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 7. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- ***uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SQ***  
Specifies the division factor for SAI1 clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15. This parameter will be used only when PLLI2S is selected as Clock Source SAI

### 55.1.3

#### RCC\_PLLSAIInitTypeDef

***RCC\_PLLSAIInitTypeDef*** is defined in the stm32f4xx\_hal\_rcc\_ex.h

##### Data Fields

- ***uint32\_t PLLSAIN***
- ***uint32\_t PLLSAIP***
- ***uint32\_t PLLSAIQ***
- ***uint32\_t PLLSAIR***

##### Field Documentation

- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIN***  
Specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between Min\_Data = 50 and Max\_Data = 432. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIP***  
Specifies division factor for OTG FS and SDIO clocks. This parameter is only available in STM32F469xx/STM32F479xx devices. This parameter must be a value of [RCCEX\\_PLLSAIP\\_Clock\\_Divider](#)
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIQ***  
Specifies the division factor for SAI1 clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIR***  
specifies the division factor for LTDC clock This parameter must be a number between Min\_Data = 2 and Max\_Data = 7. This parameter will be used only when PLLSAI is selected as Clock Source LTDC

### 55.1.4

#### RCC\_PeriphCLKInitTypeDef

***RCC\_PeriphCLKInitTypeDef*** is defined in the stm32f4xx\_hal\_rcc\_ex.h

##### Data Fields

- ***uint32\_t PeriphClockSelection***
- ***RCC\_PLLI2SInitTypeDef PLLI2S***
- ***RCC\_PLLSAIInitTypeDef PLLSAI***
- ***uint32\_t PLLI2SDivQ***
- ***uint32\_t PLLSAIDivQ***
- ***uint32\_t PLLSAIDivR***
- ***uint32\_t RTCClockSelection***
- ***uint8\_t TIMPresSelection***
- ***uint32\_t Clk48ClockSelection***
- ***uint32\_t SdioClockSelection***

##### Field Documentation

- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PeriphClockSelection***  
The Extended Clock to be configured. This parameter can be a value of [RCCEX\\_Periph\\_Clock\\_Selection](#)
- ***RCC\_PLLI2SInitTypeDef RCC\_PeriphCLKInitTypeDef::PLLI2S***  
PLL I2S structure parameters. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- ***RCC\_PLLSAIInitTypeDef RCC\_PeriphCLKInitTypeDef::PLLSAI***  
PLL SAI structure parameters. This parameter will be used only when PLLI2S is selected as Clock Source SAI or LTDC

- **`uint32_t RCC_PeriphCLKInitTypeDef::PLLI2SDivQ`**  
Specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between `Min_Data = 1` and `Max_Data = 32`. This parameter will be used only when PLLI2S is selected as Clock Source SAI.
- **`uint32_t RCC_PeriphCLKInitTypeDef::PLLSAIDivQ`**  
Specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between `Min_Data = 1` and `Max_Data = 32`. This parameter will be used only when PLLSAI is selected as Clock Source SAI.
- **`uint32_t RCC_PeriphCLKInitTypeDef::PLLSAIDivR`**  
Specifies the PLLSAI division factor for LTDC clock. This parameter must be one value of [RCCEx\\_PLLSAI\\_DIVR](#).
- **`uint32_t RCC_PeriphCLKInitTypeDef::RTCclockSelection`**  
Specifies RTC Clock Prescalers Selection. This parameter can be a value of [RCC\\_RTC\\_Clock\\_Source](#).
- **`uint8_t RCC_PeriphCLKInitTypeDef::TIMPresSelection`**  
Specifies TIM Clock Prescalers Selection. This parameter can be a value of [RCCEx\\_TIM\\_PRescaler\\_Selection](#).
- **`uint32_t RCC_PeriphCLKInitTypeDef::Clk48ClockSelection`**  
Specifies CLK48 Clock Selection this clock used OTG FS, SDIO and RNG clocks. This parameter can be a value of [RCCEx\\_CLK48\\_Clock\\_Source](#).
- **`uint32_t RCC_PeriphCLKInitTypeDef::SdioClockSelection`**  
Specifies SDIO Clock Source Selection. This parameter can be a value of [RCCEx\\_SDIO\\_Clock\\_Source](#).

## 55.2 RCCEx Firmware driver API description

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

### 55.2.1 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) and `RCC_BDCR` register are set to their reset values.

This section contains the following APIs:

- [HAL\\_RCCEx\\_PeriphCLKConfig\(\)](#)
- [HAL\\_RCCEx\\_GetPeriphCLKConfig\(\)](#)
- [HAL\\_RCCEx\\_GetPeriphCLKFreq\(\)](#)
- [HAL\\_RCCEx\\_SelectLSEMode\(\)](#)
- [HAL\\_RCCEx\\_EnablePLLI2S\(\)](#)
- [HAL\\_RCCEx\\_DisablePLLI2S\(\)](#)
- [HAL\\_RCCEx\\_EnablePLLSAI\(\)](#)
- [HAL\\_RCCEx\\_DisablePLLSAI\(\)](#)

### 55.2.2 Detailed description of functions

#### HAL\_RCCEx\_PeriphCLKConfig

##### Function name

`HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)`

##### Function description

Initializes the RCC extended peripherals clocks according to the specified parameters in the `RCC_PeriphCLKInitTypeDef`.

##### Parameters

- **PeriphClkInit:** pointer to an `RCC_PeriphCLKInitTypeDef` structure that contains the configuration information for the Extended Peripherals clocks (I2S, SAI, LTDC, RTC and TIM).



**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 Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC\_BDCR register are set to their reset values.

**HAL\_RCCEX\_GetPeriphCLKConfig**
**Function name**

```
void HAL_RCCEX_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
```

**Function description**

Configures the RCC\_PeriphCLKInitTypeDef according to the internal RCC configuration registers.

**Parameters**

- **PeriphClkInit:** pointer to an RCC\_PeriphCLKInitTypeDef structure that will be configured.

**Return values**

- **None:**

**HAL\_RCCEX\_GetPeriphCLKFreq**
**Function name**

```
uint32_t HAL_RCCEX_GetPeriphCLKFreq (uint32_t PeriphClk)
```

**Function description**

Return the peripheral clock frequency for a given peripheral(SAI..)

**Parameters**

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC\_PERIPHCLK\_I2S: I2S peripheral clock

**Return values**

- **Frequency:** in KHz

**Notes**

- Return 0 if peripheral clock identifier not managed by this API

**HAL\_RCCEX\_SelectLSEMode**
**Function name**

```
void HAL_RCCEX_SelectLSEMode (uint8_t Mode)
```

**Function description**

Select LSE mode.

**Parameters**

- **Mode:** specifies the LSE mode. This parameter can be one of the following values:
  - RCC\_LSE\_LOWPOWER\_MODE: LSE oscillator in low power mode selection
  - RCC\_LSE\_HIGHDRIVE\_MODE: LSE oscillator in High Drive mode selection

**Return values**

- **None:**

**Notes**

- This mode is only available for STM32F410xx/STM32F411xx/STM32F446xx/STM32F469xx/STM32F479xx/STM32F412Zx/STM32F412Vx/STM32F412Rx/STM32F412Cx devices.

**HAL\_RCCEX\_EnablePLL12S**
**Function name**

**HAL\_StatusTypeDef HAL\_RCCEX\_EnablePLL12S (RCC\_PLL12SInitTypeDef \* PLL12SInit)**

**Function description**

Enable PLL12S.

**Parameters**

- **PLL12SInit:** pointer to an RCC\_PLL12SInitTypeDef structure that contains the configuration information for the PLL12S

**Return values**

- **HAL:** status

**HAL\_RCCEX\_DisablePLL12S**
**Function name**

**HAL\_StatusTypeDef HAL\_RCCEX\_DisablePLL12S (void )**

**Function description**

Disable PLL12S.

**Return values**

- **HAL:** status

**HAL\_RCCEX\_EnablePLL5SAI**
**Function name**

**HAL\_StatusTypeDef HAL\_RCCEX\_EnablePLL5SAI (RCC\_PLL5SAIInitTypeDef \* PLL5SAIInit)**

**Function description**

Enable PLL5SAI.

**Parameters**

- **PLL5SAIInit:** pointer to an RCC\_PLL5SAIInitTypeDef structure that contains the configuration information for the PLL5SAI

**Return values**

- **HAL:** status

**HAL\_RCCEX\_DisablePLL5SAI**
**Function name**

**HAL\_StatusTypeDef HAL\_RCCEX\_DisablePLL5SAI (void )**

**Function description**

Disable PLL5SAI.

**Return values**

- **HAL:** status

## 55.3 RCCEX Firmware driver defines

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

### 55.3.1 RCCEX

RCCEX  
*AHB1 Peripheral Clock Enable Disable*

`__HAL_RCC_BKPSRAM_CLK_ENABLE`

`__HAL_RCC_CCMDATARAMEN_CLK_ENABLE`

`__HAL_RCC_CRC_CLK_ENABLE`

`__HAL_RCC_GPIOD_CLK_ENABLE`

`__HAL_RCC_GPIOE_CLK_ENABLE`

`__HAL_RCC_GPIOI_CLK_ENABLE`

`__HAL_RCC_GPIOF_CLK_ENABLE`

`__HAL_RCC_GPIOG_CLK_ENABLE`

`__HAL_RCC_GPIOJ_CLK_ENABLE`

`__HAL_RCC_GPIOK_CLK_ENABLE`

`__HAL_RCC_DMA2D_CLK_ENABLE`

`__HAL_RCC_ETHMAC_CLK_ENABLE`

`__HAL_RCC_ETHMACTX_CLK_ENABLE`

`__HAL_RCC_ETHMACRX_CLK_ENABLE`

`__HAL_RCC_ETHMACPTP_CLK_ENABLE`

`__HAL_RCC_USB_OTG_HS_CLK_ENABLE`

`__HAL_RCC_USB_OTG_HS_ULPI_CLK_ENABLE`

`__HAL_RCC_GPIOD_CLK_DISABLE`

`__HAL_RCC_GPIOE_CLK_DISABLE`

`__HAL_RCC_GPIOF_CLK_DISABLE`

`__HAL_RCC_GPIOG_CLK_DISABLE`

`__HAL_RCC_GPIOI_CLK_DISABLE`

`__HAL_RCC_GPIOJ_CLK_DISABLE`

`__HAL_RCC_GPIOK_CLK_DISABLE`

`__HAL_RCC_DMA2D_CLK_DISABLE`

`__HAL_RCC_ETHMAC_CLK_DISABLE`

```
__HAL_RCC_ETHMACTX_CLK_DISABLE
__HAL_RCC_ETHMACRX_CLK_DISABLE
__HAL_RCC_ETHMACPTP_CLK_DISABLE
__HAL_RCC_USB_OTG_HS_CLK_DISABLE
__HAL_RCC_USB_OTG_HS_ULPI_CLK_DISABLE
__HAL_RCC_BKPSRAM_CLK_DISABLE
__HAL_RCC_CCMDATARAMEN_CLK_DISABLE
__HAL_RCC_CRC_CLK_DISABLE
__HAL_RCC_ETH_CLK_ENABLE
__HAL_RCC_ETH_CLK_DISABLE
    AHB1 Force Release Reset
__HAL_RCC_GPIOD_FORCE_RESET
__HAL_RCC_GPIOE_FORCE_RESET
__HAL_RCC_GPIOF_FORCE_RESET
__HAL_RCC_GPIOG_FORCE_RESET
__HAL_RCC_GPIOI_FORCE_RESET
__HAL_RCC_ETHMAC_FORCE_RESET
__HAL_RCC_USB_OTG_HS_FORCE_RESET
__HAL_RCC_GPIOJ_FORCE_RESET
__HAL_RCC_GPIOK_FORCE_RESET
__HAL_RCC_DMA2D_FORCE_RESET
__HAL_RCC_CRC_FORCE_RESET
__HAL_RCC_GPIOD_RELEASE_RESET
__HAL_RCC_GPIOE_RELEASE_RESET
__HAL_RCC_GPIOF_RELEASE_RESET
__HAL_RCC_GPIOG_RELEASE_RESET
__HAL_RCC_GPIOI_RELEASE_RESET
__HAL_RCC_ETHMAC_RELEASE_RESET
__HAL_RCC_USB_OTG_HS_RELEASE_RESET
```

\_\_HAL\_RCC\_GPIOJ\_RELEASE\_RESET

\_\_HAL\_RCC\_GPIOK\_RELEASE\_RESET

\_\_HAL\_RCC\_DMA2D\_RELEASE\_RESET

\_\_HAL\_RCC\_CRC\_RELEASE\_RESET

***AHB1 Peripheral Low Power Enable Disable***

\_\_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\_GPIOI\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SRAM2\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ETHMAC\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ETHMACTX\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ETHMACRX\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ETHMACPTP\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USB\_OTG\_HS\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_GPIOJ\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_GPIOK\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SRAM3\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_DMA2D\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_CRC\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_FLITF\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SRAM1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_BKPSRAM\_CLK\_SLEEP\_ENABLE

\_\_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_GPIOI_CLK_SLEEP_DISABLE`  
`__HAL_RCC_SRAM2_CLK_SLEEP_DISABLE`  
`__HAL_RCC_ETHMAC_CLK_SLEEP_DISABLE`  
`__HAL_RCC_ETHMACTX_CLK_SLEEP_DISABLE`  
`__HAL_RCC_ETHMACRX_CLK_SLEEP_DISABLE`  
`__HAL_RCC_ETHMACPTP_CLK_SLEEP_DISABLE`  
`__HAL_RCC_USB_OTG_HS_CLK_SLEEP_DISABLE`  
`__HAL_RCC_USB_OTG_HS_ULPI_CLK_SLEEP_DISABLE`  
`__HAL_RCC_GPIOJ_CLK_SLEEP_DISABLE`  
`__HAL_RCC_GPIOK_CLK_SLEEP_DISABLE`  
`__HAL_RCC_DMA2D_CLK_SLEEP_DISABLE`  
`__HAL_RCC_CRC_CLK_SLEEP_DISABLE`  
`__HAL_RCC_FLITF_CLK_SLEEP_DISABLE`  
`__HAL_RCC_SRAM1_CLK_SLEEP_DISABLE`  
`__HAL_RCC_BKPSRAM_CLK_SLEEP_DISABLE`

***AHB1 Peripheral Clock Enable Disable Status***

`__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_GPIOI_IS_CLK_ENABLED`  
`__HAL_RCC_GPIOJ_IS_CLK_ENABLED`  
`__HAL_RCC_GPIOK_IS_CLK_ENABLED`  
`__HAL_RCC_DMA2D_IS_CLK_ENABLED`  
`__HAL_RCC_ETHMAC_IS_CLK_ENABLED`  
`__HAL_RCC_ETHMACTX_IS_CLK_ENABLED`  
`__HAL_RCC_ETHMACRX_IS_CLK_ENABLED`  
`__HAL_RCC_ETHMACPTP_IS_CLK_ENABLED`  
`__HAL_RCC_USB_OTG_HS_IS_CLK_ENABLED`

```
__HAL_RCC_USB_OTG_HS_ULPI_IS_CLK_ENABLED
__HAL_RCC_BKPSRAM_IS_CLK_ENABLED
__HAL_RCC_CCMDATARAMEN_IS_CLK_ENABLED
__HAL_RCC_CRC_IS_CLK_ENABLED
__HAL_RCC_ETH_IS_CLK_ENABLED
__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_GPIOI_IS_CLK_DISABLED
__HAL_RCC_GPIOJ_IS_CLK_DISABLED
__HAL_RCC_GPIOK_IS_CLK_DISABLED
__HAL_RCC_DMA2D_IS_CLK_DISABLED
__HAL_RCC_ETHMAC_IS_CLK_DISABLED
__HAL_RCC_ETHMACTX_IS_CLK_DISABLED
__HAL_RCC_ETHMACRX_IS_CLK_DISABLED
__HAL_RCC_ETHMACPTP_IS_CLK_DISABLED
__HAL_RCC_USB_OTG_HS_IS_CLK_DISABLED
__HAL_RCC_USB_OTG_HS_ULPI_IS_CLK_DISABLED
__HAL_RCC_BKPSRAM_IS_CLK_DISABLED
__HAL_RCC_CCMDATARAMEN_IS_CLK_DISABLED
__HAL_RCC_CRC_IS_CLK_DISABLED
__HAL_RCC_ETH_IS_CLK_DISABLED
    AHB2 Peripheral Clock Enable Disable
__HAL_RCC_DCM1_CLK_ENABLE
__HAL_RCC_DCM1_CLK_DISABLE
__HAL_RCC_Cryp_CLK_ENABLE
__HAL_RCC_HASH_CLK_ENABLE
__HAL_RCC_Cryp_CLK_DISABLE
```

\_\_HAL\_RCC\_HASH\_CLK\_DISABLE

\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_ENABLE

\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_DISABLE

\_\_HAL\_RCC\_RNG\_CLK\_ENABLE

\_\_HAL\_RCC\_RNG\_CLK\_DISABLE

***AHB2 Force Release Reset***

\_\_HAL\_RCC\_AHB2\_FORCE\_RESET

\_\_HAL\_RCC\_USB\_OTG\_FS\_FORCE\_RESET

\_\_HAL\_RCC\_RNG\_FORCE\_RESET

\_\_HAL\_RCC\_DCMI\_FORCE\_RESET

\_\_HAL\_RCC\_AHB2\_RELEASE\_RESET

\_\_HAL\_RCC\_USB\_OTG\_FS\_RELEASE\_RESET

\_\_HAL\_RCC\_RNG\_RELEASE\_RESET

\_\_HAL\_RCC\_DCMI\_RELEASE\_RESET

\_\_HAL\_RCC\_Cryp\_FORCE\_RESET

\_\_HAL\_RCC\_HASH\_FORCE\_RESET

\_\_HAL\_RCC\_Cryp\_RELEASE\_RESET

\_\_HAL\_RCC\_HASH\_RELEASE\_RESET

***AHB2 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_RNG\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_RNG\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_DCMI\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_DCMI\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_Cryp\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_HASH\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_Cryp\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_HASH\_CLK\_SLEEP\_DISABLE

***AHB2 Peripheral Clock Enable Disable Status***



\_\_HAL\_RCC\_DCMIS\_CLK\_ENABLED

\_\_HAL\_RCC\_DCMIS\_CLK\_DISABLED

\_\_HAL\_RCC\_CRYPIS\_CLK\_ENABLED

\_\_HAL\_RCC\_CRYPIS\_CLK\_DISABLED

\_\_HAL\_RCC\_HASHIS\_CLK\_ENABLED

\_\_HAL\_RCC\_HASHIS\_CLK\_DISABLED

\_\_HAL\_RCC\_USB\_OTG\_FSIS\_CLK\_ENABLED

\_\_HAL\_RCC\_USB\_OTG\_FSIS\_CLK\_DISABLED

\_\_HAL\_RCC\_RNGIS\_CLK\_ENABLED

\_\_HAL\_RCC\_RNGIS\_CLK\_DISABLED

***AHB3 Peripheral Clock Enable Disable***

\_\_HAL\_RCC\_FMC\_CLK\_ENABLE

\_\_HAL\_RCC\_FMC\_CLK\_DISABLE

\_\_HAL\_RCC\_QSPI\_CLK\_ENABLE

\_\_HAL\_RCC\_QSPI\_CLK\_DISABLE

***AHB3 Force Release Reset***

\_\_HAL\_RCC\_AHB3\_FORCE\_RESET

\_\_HAL\_RCC\_AHB3\_RELEASE\_RESET

\_\_HAL\_RCC\_FMC\_FORCE\_RESET

\_\_HAL\_RCC\_FMC\_RELEASE\_RESET

\_\_HAL\_RCC\_QSPI\_FORCE\_RESET

\_\_HAL\_RCC\_QSPI\_RELEASE\_RESET

***AHB3 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_FMC\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_FMC\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_QSPI\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_QSPI\_CLK\_SLEEP\_DISABLE

***AHB3 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_FMC\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_FMC\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_QSPI\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_QSPI\_IS\_CLK\_DISABLED

***APB1 Peripheral Clock Enable Disable***

\_\_HAL\_RCC\_TIM6\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM7\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM12\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM13\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM14\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM14\_CLK\_ENABLE

\_\_HAL\_RCC\_USART3\_CLK\_ENABLE

\_\_HAL\_RCC\_UART4\_CLK\_ENABLE

\_\_HAL\_RCC\_UART5\_CLK\_ENABLE

\_\_HAL\_RCC\_CAN1\_CLK\_ENABLE

\_\_HAL\_RCC\_CAN2\_CLK\_ENABLE

\_\_HAL\_RCC\_DAC\_CLK\_ENABLE

\_\_HAL\_RCC\_UART7\_CLK\_ENABLE

\_\_HAL\_RCC\_UART8\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM2\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM3\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM4\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI3\_CLK\_ENABLE

\_\_HAL\_RCC\_I2C3\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM2\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM3\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM4\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI3\_CLK\_DISABLE

\_\_HAL\_RCC\_I2C3\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM6\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM7\_CLK\_DISABLE

---

`__HAL_RCC_TIM12_CLK_DISABLE`  
`__HAL_RCC_TIM13_CLK_DISABLE`  
`__HAL_RCC_TIM14_CLK_DISABLE`  
`__HAL_RCC_USART3_CLK_DISABLE`  
`__HAL_RCC_UART4_CLK_DISABLE`  
`__HAL_RCC_UART5_CLK_DISABLE`  
`__HAL_RCC_CAN1_CLK_DISABLE`  
`__HAL_RCC_CAN2_CLK_DISABLE`  
`__HAL_RCC_DAC_CLK_DISABLE`  
`__HAL_RCC_UART7_CLK_DISABLE`  
`__HAL_RCC_UART8_CLK_DISABLE`

***APB1 Force Release Reset***

`__HAL_RCC_TIM6_FORCE_RESET`  
`__HAL_RCC_TIM7_FORCE_RESET`  
`__HAL_RCC_TIM12_FORCE_RESET`  
`__HAL_RCC_TIM13_FORCE_RESET`  
`__HAL_RCC_TIM14_FORCE_RESET`  
`__HAL_RCC_USART3_FORCE_RESET`  
`__HAL_RCC_UART4_FORCE_RESET`  
`__HAL_RCC_UART5_FORCE_RESET`  
`__HAL_RCC_CAN1_FORCE_RESET`  
`__HAL_RCC_CAN2_FORCE_RESET`  
`__HAL_RCC_DAC_FORCE_RESET`  
`__HAL_RCC_UART7_FORCE_RESET`  
`__HAL_RCC_UART8_FORCE_RESET`  
`__HAL_RCC_TIM2_FORCE_RESET`  
`__HAL_RCC_TIM3_FORCE_RESET`  
`__HAL_RCC_TIM4_FORCE_RESET`  
`__HAL_RCC_SPI3_FORCE_RESET`

\_\_HAL\_RCC\_I2C3\_FORCE\_RESET

\_\_HAL\_RCC\_TIM2\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM3\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM4\_RELEASE\_RESET

\_\_HAL\_RCC\_SPI3\_RELEASE\_RESET

\_\_HAL\_RCC\_I2C3\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM6\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM7\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM12\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM13\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM14\_RELEASE\_RESET

\_\_HAL\_RCC\_USART3\_RELEASE\_RESET

\_\_HAL\_RCC\_UART4\_RELEASE\_RESET

\_\_HAL\_RCC\_UART5\_RELEASE\_RESET

\_\_HAL\_RCC\_CAN1\_RELEASE\_RESET

\_\_HAL\_RCC\_CAN2\_RELEASE\_RESET

\_\_HAL\_RCC\_DAC\_RELEASE\_RESET

\_\_HAL\_RCC\_UART7\_RELEASE\_RESET

\_\_HAL\_RCC\_UART8\_RELEASE\_RESET

***APB1 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_TIM6\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM7\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM12\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM13\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_TIM14\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_USART3\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_UART4\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_UART5\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_CAN1\_CLK\_SLEEP\_ENABLE

```
__HAL_RCC_CAN2_CLK_SLEEP_ENABLE
__HAL_RCC_DAC_CLK_SLEEP_ENABLE
__HAL_RCC_UART7_CLK_SLEEP_ENABLE
__HAL_RCC_UART8_CLK_SLEEP_ENABLE
__HAL_RCC_TIM2_CLK_SLEEP_ENABLE
__HAL_RCC_TIM3_CLK_SLEEP_ENABLE
__HAL_RCC_TIM4_CLK_SLEEP_ENABLE
__HAL_RCC_SPI3_CLK_SLEEP_ENABLE
__HAL_RCC_I2C3_CLK_SLEEP_ENABLE
__HAL_RCC_TIM2_CLK_SLEEP_DISABLE
__HAL_RCC_TIM3_CLK_SLEEP_DISABLE
__HAL_RCC_TIM4_CLK_SLEEP_DISABLE
__HAL_RCC_SPI3_CLK_SLEEP_DISABLE
__HAL_RCC_I2C3_CLK_SLEEP_DISABLE
__HAL_RCC_TIM6_CLK_SLEEP_DISABLE
__HAL_RCC_TIM7_CLK_SLEEP_DISABLE
__HAL_RCC_TIM12_CLK_SLEEP_DISABLE
__HAL_RCC_TIM13_CLK_SLEEP_DISABLE
__HAL_RCC_TIM14_CLK_SLEEP_DISABLE
__HAL_RCC_USART3_CLK_SLEEP_DISABLE
__HAL_RCC_UART4_CLK_SLEEP_DISABLE
__HAL_RCC_UART5_CLK_SLEEP_DISABLE
__HAL_RCC_CAN1_CLK_SLEEP_DISABLE
__HAL_RCC_CAN2_CLK_SLEEP_DISABLE
__HAL_RCC_DAC_CLK_SLEEP_DISABLE
__HAL_RCC_UART7_CLK_SLEEP_DISABLE
__HAL_RCC_UART8_CLK_SLEEP_DISABLE
    APB1 Peripheral Clock Enable Disable Status
__HAL_RCC_TIM2_IS_CLK_ENABLED
```

```
__HAL_RCC_TIM3_IS_CLK_ENABLED
__HAL_RCC_TIM4_IS_CLK_ENABLED
__HAL_RCC_SPI3_IS_CLK_ENABLED
__HAL_RCC_I2C3_IS_CLK_ENABLED
__HAL_RCC_TIM6_IS_CLK_ENABLED
__HAL_RCC_TIM7_IS_CLK_ENABLED
__HAL_RCC_TIM12_IS_CLK_ENABLED
__HAL_RCC_TIM13_IS_CLK_ENABLED
__HAL_RCC_TIM14_IS_CLK_ENABLED
__HAL_RCC_USART3_IS_CLK_ENABLED
__HAL_RCC_UART4_IS_CLK_ENABLED
__HAL_RCC_UART5_IS_CLK_ENABLED
__HAL_RCC_CAN1_IS_CLK_ENABLED
__HAL_RCC_CAN2_IS_CLK_ENABLED
__HAL_RCC_DAC_IS_CLK_ENABLED
__HAL_RCC_UART7_IS_CLK_ENABLED
__HAL_RCC_UART8_IS_CLK_ENABLED
__HAL_RCC_TIM2_IS_CLK_DISABLED
__HAL_RCC_TIM3_IS_CLK_DISABLED
__HAL_RCC_TIM4_IS_CLK_DISABLED
__HAL_RCC_SPI3_IS_CLK_DISABLED
__HAL_RCC_I2C3_IS_CLK_DISABLED
__HAL_RCC_TIM6_IS_CLK_DISABLED
__HAL_RCC_TIM7_IS_CLK_DISABLED
__HAL_RCC_TIM12_IS_CLK_DISABLED
__HAL_RCC_TIM13_IS_CLK_DISABLED
__HAL_RCC_TIM14_IS_CLK_DISABLED
__HAL_RCC_USART3_IS_CLK_DISABLED
```

\_\_HAL\_RCC\_UART4\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_UART5\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_CAN1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_CAN2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_DAC\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_UART7\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_UART8\_IS\_CLK\_DISABLED

***APB2 Peripheral Clock Enable Disable***

\_\_HAL\_RCC\_TIM8\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC2\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC3\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI5\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI6\_CLK\_ENABLE

\_\_HAL\_RCC\_SAI1\_CLK\_ENABLE

\_\_HAL\_RCC\_SDIO\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI4\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM10\_CLK\_ENABLE

\_\_HAL\_RCC\_SDIO\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI4\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM10\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM8\_CLK\_DISABLE

\_\_HAL\_RCC\_ADC2\_CLK\_DISABLE

\_\_HAL\_RCC\_ADC3\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI5\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI6\_CLK\_DISABLE

\_\_HAL\_RCC\_SAI1\_CLK\_DISABLE

\_\_HAL\_RCC\_LTDC\_CLK\_ENABLE

\_\_HAL\_RCC\_LTDC\_CLK\_DISABLE

\_\_HAL\_RCC\_DSI\_CLK\_ENABLE

\_\_HAL\_RCC\_DSI\_CLK\_DISABLE

***APB2 Force Release Reset***

\_\_HAL\_RCC\_TIM8\_FORCE\_RESET

\_\_HAL\_RCC\_SPI5\_FORCE\_RESET

\_\_HAL\_RCC\_SPI6\_FORCE\_RESET

\_\_HAL\_RCC\_SAI1\_FORCE\_RESET

\_\_HAL\_RCC\_SDIO\_FORCE\_RESET

\_\_HAL\_RCC\_SPI4\_FORCE\_RESET

\_\_HAL\_RCC\_TIM10\_FORCE\_RESET

\_\_HAL\_RCC\_SDIO\_RELEASE\_RESET

\_\_HAL\_RCC\_SPI4\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM10\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM8\_RELEASE\_RESET

\_\_HAL\_RCC\_SPI5\_RELEASE\_RESET

\_\_HAL\_RCC\_SPI6\_RELEASE\_RESET

\_\_HAL\_RCC\_SAI1\_RELEASE\_RESET

\_\_HAL\_RCC\_LTDC\_FORCE\_RESET

\_\_HAL\_RCC\_LTDC\_RELEASE\_RESET

\_\_HAL\_RCC\_DSI\_FORCE\_RESET

\_\_HAL\_RCC\_DSI\_RELEASE\_RESET

***APB2 Peripheral Low Power Enable Disable***

\_\_HAL\_RCC\_TIM8\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ADC2\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_ADC3\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SPI5\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SPI6\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SAI1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SDIO\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_SPI4\_CLK\_SLEEP\_ENABLE



`__HAL_RCC_TIM10_CLK_SLEEP_ENABLE`  
`__HAL_RCC_SDIO_CLK_SLEEP_DISABLE`  
`__HAL_RCC_SPI4_CLK_SLEEP_DISABLE`  
`__HAL_RCC_TIM10_CLK_SLEEP_DISABLE`  
`__HAL_RCC_TIM8_CLK_SLEEP_DISABLE`  
`__HAL_RCC_ADC2_CLK_SLEEP_DISABLE`  
`__HAL_RCC_ADC3_CLK_SLEEP_DISABLE`  
`__HAL_RCC_SPI5_CLK_SLEEP_DISABLE`  
`__HAL_RCC_SPI6_CLK_SLEEP_DISABLE`  
`__HAL_RCC_SAI1_CLK_SLEEP_DISABLE`  
`__HAL_RCC_LTDC_CLK_SLEEP_ENABLE`  
`__HAL_RCC_LTDC_CLK_SLEEP_DISABLE`  
`__HAL_RCC_DSI_CLK_SLEEP_ENABLE`  
`__HAL_RCC_DSI_CLK_SLEEP_DISABLE`

***APB2 Peripheral Clock Enable Disable Status***

`__HAL_RCC_TIM8_IS_CLK_ENABLED`  
`__HAL_RCC_ADC2_IS_CLK_ENABLED`  
`__HAL_RCC_ADC3_IS_CLK_ENABLED`  
`__HAL_RCC_SPI5_IS_CLK_ENABLED`  
`__HAL_RCC_SPI6_IS_CLK_ENABLED`  
`__HAL_RCC_SAI1_IS_CLK_ENABLED`  
`__HAL_RCC_SDIO_IS_CLK_ENABLED`  
`__HAL_RCC_SPI4_IS_CLK_ENABLED`  
`__HAL_RCC_TIM10_IS_CLK_ENABLED`  
`__HAL_RCC_SDIO_IS_CLK_DISABLED`  
`__HAL_RCC_SPI4_IS_CLK_DISABLED`  
`__HAL_RCC_TIM10_IS_CLK_DISABLED`  
`__HAL_RCC_TIM8_IS_CLK_DISABLED`  
`__HAL_RCC_ADC2_IS_CLK_DISABLED`

\_\_HAL\_RCC\_ADC3\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SPI5\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SPI6\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SAI1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_LTDC\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_LTDC\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_DSI\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_DSI\_IS\_CLK\_DISABLED

***RCC BitAddress AliasRegion***

RCC\_PLLSAION\_BIT\_NUMBER

RCC\_CR\_PLLSAION\_BB

PLLSAI\_TIMEOUT\_VALUE

RCC\_PLLI2SON\_BIT\_NUMBER

RCC\_CR\_PLLI2SON\_BB

RCC\_DCKCFGR\_OFFSET

RCC\_TIMPRE\_BIT\_NUMBER

RCC\_DCKCFGR\_TIMPRE\_BB

RCC\_CFGR\_OFFSET

RCC\_I2SSRC\_BIT\_NUMBER

RCC\_CFGR\_I2SSRC\_BB

PLLI2S\_TIMEOUT\_VALUE

PLL\_TIMEOUT\_VALUE

***RCC CLK48 Clock Source***

RCC\_CLK48CLKSOURCE\_PLLQ

RCC\_CLK48CLKSOURCE\_PLLSAIP

***RCC DSI Clock Source***

RCC\_DSICLKSOURCE\_DSIPHY

RCC\_DSICLKSOURCE\_PLLR

***RCCEX Exported Macros***

## **\_\_HAL\_RCC\_PLL\_CONFIG**

### **Description:**

- Macro to configure the main PLL clock source, multiplication and division factors.

### **Parameters:**

- **\_\_RCC\_PLLSource\_\_**: specifies the PLL entry clock source. This parameter can be one of the following values:
  - **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 number between **Min\_Data = 2** and **Max\_Data = 63**.
- **\_\_PLLN\_\_**: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between **Min\_Data = 50** and **Max\_Data = 432**.
- **\_\_PLL\_P\_\_**: specifies the division factor for main system clock (SYSCLK) This parameter must be a number in the range {2, 4, 6, or 8}.
- **\_\_PLLQ\_\_**: specifies the division factor for OTG FS, SDIO and RNG clocks This parameter must be a number between **Min\_Data = 2** and **Max\_Data = 15**.
- **\_\_PLLR\_\_**: PLL division factor for I2S, SAI, SYSTEM, SPDIFRX clocks. This parameter must be a number between **Min\_Data = 2** and **Max\_Data = 7**.

### **Notes:**

- This function must be used only when the main PLL is disabled.
- This clock source (**RCC\_PLLSource**) is common for the main PLL and **PLLI2S**.
- You have to set the **PLLM** parameter correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.
- You have to set the **PLLN** parameter correctly to ensure that the VCO output frequency is between 100 and 432 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 SDIO and RNG need a frequency lower than or equal to 48 MHz to work correctly.
- This parameter is only available in STM32F446xx/STM32F469xx/STM32F479xx/ STM32F412Zx/ STM32F412Vx/STM32F412Rx/STM32F412Cx/STM32F413xx/STM32F423xx devices.

## **\_\_HAL\_RCC\_PLLI2S\_ENABLE**

### **Notes:**

- The **PLLI2S** is disabled by hardware when entering STOP and STANDBY modes.

## **\_\_HAL\_RCC\_PLLI2S\_DISABLE**

## **\_\_HAL\_RCC\_PLLI2S\_CONFIG**

### **Description:**

- Macro to configure the **PLLI2S** clock multiplication and division factors .

### **Parameters:**

- **\_\_PLLI2SN\_\_**: specifies the multiplication factor for **PLLI2S** VCO output clock This parameter must be a number between **Min\_Data = 50** and **Max\_Data = 432**.
- **\_\_PLLI2SR\_\_**: specifies the division factor for I2S clock This parameter must be a number between **Min\_Data = 2** and **Max\_Data = 7**.

### **Notes:**

- This macro must be used only when the **PLLI2S** is disabled. **PLLI2S** clock source is common with the main PLL (configured in **HAL\_RCC\_ClockConfig()** API).
- You have to set the **PLLI2SN** parameter correctly to ensure that the VCO output frequency is between **Min\_Data = 100** and **Max\_Data = 432** MHz.
- You have to set the **PLLI2SR** parameter correctly to not exceed 192 MHz on the I2S clock frequency.

### **\_\_HAL\_RCC\_PLLI2S\_SAICLK\_CONFIG**

**Description:**

- Macro used by the SAI HAL driver to configure the PLLI2S clock multiplication and division factors.

**Parameters:**

- `__PLLI2SN__`: specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between `Min_Data = 50` and `Max_Data = 432`.
- `__PLLI2SQ__`: specifies the division factor for SAI1 clock. This parameter must be a number between `Min_Data = 2` and `Max_Data = 15`.
- `__PLLI2SR__`: specifies the division factor for I2S clock This parameter must be a number between `Min_Data = 2` and `Max_Data = 7`.

**Notes:**

- This macro must be used only when the PLLI2S is disabled. PLLI2S clock source is common with the main PLL (configured in `HAL_RCC_ClockConfig()` API)
- You have to set the PLLI2SN parameter correctly to ensure that the VCO output frequency is between `Min_Data = 100` and `Max_Data = 432` MHz.
- the PLLI2SQ parameter is only available with STM32F427xx/437xx/429xx/439xx/469xx/479xx Devices and can be configured using the `__HAL_RCC_PLLI2S_PLLSAICLK_CONFIG()` macro
- You have to set the PLLI2SR parameter correctly to not exceed 192 MHz on the I2S clock frequency.

### **\_\_HAL\_RCC\_PLLSAI\_ENABLE**

**Notes:**

- The PLLSAI is only available with STM32F429x/439x Devices. The PLLSAI is disabled by hardware when entering STOP and STANDBY modes.

### **\_\_HAL\_RCC\_PLLSAI\_DISABLE**

### **\_\_HAL\_RCC\_PLLSAI\_CONFIG**

**Description:**

- Macro to configure the PLLSAI clock multiplication and division factors.

**Parameters:**

- `__PLLSAIN__`: specifies the multiplication factor for PLLSAI VCO output clock. This parameter must be a number between `Min_Data = 50` and `Max_Data = 432`.
- `__PLLSAIP__`: specifies division factor for SDIO and CLK48 clocks. This parameter must be a number in the range {2, 4, 6, or 8}.
- `__PLLSAIQ__`: specifies the division factor for SAI clock This parameter must be a number between `Min_Data = 2` and `Max_Data = 15`.
- `__PLLSAIR__`: specifies the division factor for LTDC clock This parameter must be a number between `Min_Data = 2` and `Max_Data = 7`.

**Notes:**

- You have to set the PLLSAIN parameter correctly to ensure that the VCO output frequency is between `Min_Data = 100` and `Max_Data = 432` MHz.

### **\_\_HAL\_RCC\_PLLI2S\_PLLSAICLKDIVQ\_CONFIG**

**Description:**

- Macro to configure the SAI clock Divider coming from PLLI2S.

**Parameters:**

- `__PLLI2SDivQ__`: specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between 1 and 32. SAI1 clock frequency =  $f(\text{PLLI2SQ}) / \text{__PLLI2SDivQ__}$

**Notes:**

- This function must be called before enabling the PLLI2S.

### \_\_HAL\_RCC\_PLLSAI\_PLLSAICLKDIVQ\_CONFIG

**Description:**

- Macro to configure the SAI clock Divider coming from PLLSAI.

**Parameters:**

- `__PLLSAIDivQ__`: specifies the PLLSAI division factor for SAI1 clock . This parameter must be a number between `Min_Data = 1` and `Max_Data = 32`. SAI1 clock frequency =  $f(\text{PLLSAIQ}) / \text{__PLLSAIDivQ__}$

**Notes:**

- This function must be called before enabling the PLLSAI.

### \_\_HAL\_RCC\_PLLSAI\_PLLSAICLKDIVR\_CONFIG

**Description:**

- Macro to configure the LTDC clock Divider coming from PLLSAI.

**Parameters:**

- `__PLLSAIDivR__`: specifies the PLLSAI division factor for LTDC clock . This parameter must be a number between `Min_Data = 2` and `Max_Data = 16`. LTDC clock frequency =  $f(\text{PLLSAIR}) / \text{__PLLSAIDivR__}$

**Notes:**

- The LTDC peripheral is only available with STM32F427/437/429/439/469/479xx Devices. This function must be called before enabling the PLLSAI.

### \_\_HAL\_RCC\_I2S\_CONFIG

**Description:**

- Macro to configure the I2S clock source (I2SCLK).

**Parameters:**

- `__SOURCE__`: specifies the I2S clock source. This parameter can be one of the following values:
  - `RCC_I2SCLKSOURCE_PLLI2S`: PLLI2S clock used as I2S clock source.
  - `RCC_I2SCLKSOURCE_EXT`: External clock mapped on the I2S\_CKIN pin used as I2S clock source.

**Notes:**

- This function must be called before enabling the I2S APB clock.

### \_\_HAL\_RCC\_GET\_I2S\_SOURCE

**Description:**

- Macro to get the I2S clock source (I2SCLK).

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_I2SCLKSOURCE_PLLI2S`: PLLI2S clock used as I2S clock source.
  - `RCC_I2SCLKSOURCE_EXT` External clock mapped on the I2S\_CKIN pin used as I2S clock source

### \_\_HAL\_RCC\_SAI\_BLOCKACLKSOURCE\_CONFIG

**Description:**

- Macro to configure SAI1BlockA clock source selection.

**Parameters:**

- `__SOURCE__`: specifies the SAI Block A clock source. This parameter can be one of the following values:
  - `RCC_SAIACLKSOURCE_PLLI2S`: PLLI2S\_Q clock divided by PLLI2SDIVQ used as SAI1 Block A clock.
  - `RCC_SAIACLKSOURCE_PLLSAI`: PLLISAI\_Q clock divided by PLLSAIDIVQ used as SAI1 Block A clock.
  - `RCC_SAIACLKSOURCE_Ext`: External clock mapped on the I2S\_CKIN pin used as SAI1 Block A clock.

**Notes:**

- The SAI peripheral is only available with STM32F427/437/429/439/469/479xx Devices. This function must be called before enabling PLLSAI, PLLI2S and the SAI clock.

### **\_\_HAL\_RCC\_SAI\_BLOCKBCLKSOURCE\_CONFIG**

**Description:**

- Macro to configure SAI1BlockB clock source selection.

**Parameters:**

- `__SOURCE__`: specifies the SAI Block B clock source. This parameter can be one of the following values:
  - `RCC_SAIBCLKSOURCE_PLLI2S`: PLLI2S\_Q clock divided by PLLI2SDIVQ used as SAI1 Block B clock.
  - `RCC_SAIBCLKSOURCE_PLLSAI`: PLLISAI\_Q clock divided by PLLSAIDIVQ used as SAI1 Block B clock.
  - `RCC_SAIBCLKSOURCE_Ext`: External clock mapped on the I2S\_CKIN pin used as SAI1 Block B clock.

**Notes:**

- The SAI peripheral is only available with STM32F427/437/429/439/469/479xx Devices. This function must be called before enabling PLLSAI, PLLI2S and the SAI clock.

### **\_\_HAL\_RCC\_CLK48\_CONFIG**

**Description:**

- Macro to configure the CLK48 clock.

**Parameters:**

- `__SOURCE__`: specifies the CLK48 clock source. This parameter can be one of the following values:
  - `RCC_CLK48CLKSOURCE_PLLQ`: PLL VCO Output divided by PLLQ used as CLK48 clock.
  - `RCC_CLK48CLKSOURCE_PLLSAIP`: PLLSAI VCO Output divided by PLLSAIP used as CLK48 clock.

### **\_\_HAL\_RCC\_GET\_CLK48\_SOURCE**

**Description:**

- Macro to Get the CLK48 clock.

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_CLK48CLKSOURCE_PLLQ`: PLL VCO Output divided by PLLQ used as CLK48 clock.
  - `RCC_CLK48CLKSOURCE_PLLSAIP`: PLLSAI VCO Output divided by PLLSAIP used as CLK48 clock.

### **\_\_HAL\_RCC\_SDIO\_CONFIG**

**Description:**

- Macro to configure the SDIO clock.

**Parameters:**

- `__SOURCE__`: specifies the SDIO clock source. This parameter can be one of the following values:
  - `RCC_SDIOCLKSOURCE_CLK48`: CLK48 output used as SDIO clock.
  - `RCC_SDIOCLKSOURCE_SYSCLK`: System clock output used as SDIO clock.

### **\_\_HAL\_RCC\_GET\_SDIO\_SOURCE**

**Description:**

- Macro to Get the SDIO clock.

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_SDIOCLKSOURCE_CLK48`: CLK48 output used as SDIO clock.
  - `RCC_SDIOCLKSOURCE_SYSCLK`: System clock output used as SDIO clock.

### \_\_HAL\_RCC\_DSI\_CONFIG

**Description:**

- Macro to configure the DSI clock.

**Parameters:**

- `__SOURCE__`: specifies the DSI clock source. This parameter can be one of the following values:
  - `RCC_DSICLKSOURCE_PLLR`: PLLR output used as DSI clock.
  - `RCC_DSICLKSOURCE_DSIPHY`: DSI-PHY output used as DSI clock.

### \_\_HAL\_RCC\_GET\_DSI\_SOURCE

**Description:**

- Macro to Get the DSI clock.

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_DSICLKSOURCE_PLLR`: PLLR output used as DSI clock.
  - `RCC_DSICLKSOURCE_DSIPHY`: DSI-PHY output used as DSI clock.

### \_\_HAL\_RCC\_TIMCLKPRESCALER

**Description:**

- Macro to configure the Timers clocks prescalers.

**Parameters:**

- `__PRESC__`: specifies the Timers clocks prescalers selection This parameter can be one of the following values:
  - `RCC_TIMPRES_DESACTIVATED`: The Timers kernels clocks prescaler is equal to HPRE if PPREx is corresponding to division by 1 or 2, else it is equal to  $[(HPRE * PPREx) / 2]$  if PPREx is corresponding to division by 4 or more.
  - `RCC_TIMPRES_ACTIVATED`: The Timers kernels clocks prescaler is equal to HPRE if PPREx is corresponding to division by 1, 2 or 4, else it is equal to  $[(HPRE * PPREx) / 4]$  if PPREx is corresponding to division by 8 or more.

**Notes:**

- This feature is only available with STM32F429x/439x Devices.

### \_\_HAL\_RCC\_PLLSAI\_ENABLE\_IT

### \_\_HAL\_RCC\_PLLSAI\_DISABLE\_IT

### \_\_HAL\_RCC\_PLLSAI\_CLEAR\_IT

### \_\_HAL\_RCC\_PLLSAI\_GET\_IT

**Description:**

- Check the PLLSAI RDY interrupt has occurred or not.

**Return value:**

- The: new state (TRUE or FALSE).

### \_\_HAL\_RCC\_PLLSAI\_GET\_FLAG

**Description:**

- Check PLLSAI RDY flag is set or not.

**Return value:**

- The: new state (TRUE or FALSE).

**I2S Clock Source**

### RCC\_I2SCLKSOURCE\_PLLI2S

**RCC\_I2SCLKSOURCE\_EXT*****RCC Private macros to check input parameters***

IS\_RCC\_PLLN\_VALUE

IS\_RCC\_PLLI2SN\_VALUE

IS\_RCC\_PERIPHCLK

IS\_RCC\_PLLI2SR\_VALUE

IS\_RCC\_PLLI2SQ\_VALUE

IS\_RCC\_PLLSAIN\_VALUE

IS\_RCC\_PLLSAIQ\_VALUE

IS\_RCC\_PLLSAIR\_VALUE

IS\_RCC\_PLLSAI\_DIVQ\_VALUE

IS\_RCC\_PLLI2S\_DIVQ\_VALUE

IS\_RCC\_PLLSAI\_DIVR\_VALUE

IS\_RCC\_PLLR\_VALUE

IS\_RCC\_PLLSAIP\_VALUE

IS\_RCC\_CLK48CLKSOURCE

IS\_RCC\_SDIOCLKSOURCE

IS\_RCC\_DSIBYTELANECLKSOURCE

IS\_RCC\_LSE\_MODE

IS\_RCC\_MCO2SOURCE

***RCC LSE Dual Mode Selection***

RCC\_LSE\_LOWPOWER\_MODE

RCC\_LSE\_HIGHDRIVE\_MODE

***RCC Extended MCOx Clock Config***



## \_\_HAL\_RCC\_MCO1\_CONFIG

### Description:

- Macro to configure the MCO1 clock.

### Parameters:

- `__MCOCLKSOURCE__`: specifies the MCO clock source. This parameter can be one of the following values:
  - `RCC_MCO1SOURCE_HSI`: HSI clock selected as MCO1 source
  - `RCC_MCO1SOURCE_LSE`: LSE clock selected as MCO1 source
  - `RCC_MCO1SOURCE_HSE`: HSE clock selected as MCO1 source
  - `RCC_MCO1SOURCE_PLLCLK`: main PLL clock selected as MCO1 source
- `__MCODIV__`: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - `RCC_MCODIV_1`: no division applied to MCOx clock
  - `RCC_MCODIV_2`: division by 2 applied to MCOx clock
  - `RCC_MCODIV_3`: division by 3 applied to MCOx clock
  - `RCC_MCODIV_4`: division by 4 applied to MCOx clock
  - `RCC_MCODIV_5`: division by 5 applied to MCOx clock

## \_\_HAL\_RCC\_MCO2\_CONFIG

### Description:

- Macro to configure the MCO2 clock.

### Parameters:

- `__MCOCLKSOURCE__`: specifies the MCO clock source. This parameter can be one of the following values:
  - `RCC_MCO2SOURCE_SYSCLK`: System clock (SYSCLK) selected as MCO2 source
  - `RCC_MCO2SOURCE_PLLI2SCLK`: PLLI2S clock selected as MCO2 source, available for all STM32F4 devices except STM32F410xx
  - `RCC_MCO2SOURCE_I2SCLK`: I2SCLK clock selected as MCO2 source, available only for STM32F410Rx devices
  - `RCC_MCO2SOURCE_HSE`: HSE clock selected as MCO2 source
  - `RCC_MCO2SOURCE_PLLCLK`: main PLL clock selected as MCO2 source
- `__MCODIV__`: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - `RCC_MCODIV_1`: no division applied to MCOx clock
  - `RCC_MCODIV_2`: division by 2 applied to MCOx clock
  - `RCC_MCODIV_3`: division by 3 applied to MCOx clock
  - `RCC_MCODIV_4`: division by 4 applied to MCOx clock
  - `RCC_MCODIV_5`: division by 5 applied to MCOx clock

### Notes:

- For STM32F410Rx devices, to output I2SCLK clock on MCO2, you should have at least one of the SPI clocks enabled (SPI1, SPI2 or SPI5).

### **RCC Periph Clock Selection**

`RCC_PERIPHCLK_I2S`

`RCC_PERIPHCLK_SAI_PLLI2S`

`RCC_PERIPHCLK_SAI_PLLSAI`

`RCC_PERIPHCLK_LTDC`

`RCC_PERIPHCLK_TIM`

`RCC_PERIPHCLK_RTC`

RCC\_PERIPHCLK\_PLLI2S

RCC\_PERIPHCLK\_CLK48

RCC\_PERIPHCLK\_SDIO

*RCC PLLSAIP Clock Divider*

RCC\_PLLSAIP\_DIV2

RCC\_PLLSAIP\_DIV4

RCC\_PLLSAIP\_DIV6

RCC\_PLLSAIP\_DIV8

*RCC PLLSAI DIVR*

RCC\_PLLSAIDIVR\_2

RCC\_PLLSAIDIVR\_4

RCC\_PLLSAIDIVR\_8

RCC\_PLLSAIDIVR\_16

*RCC SAI BlockA Clock Source*

RCC\_SAIACLKSOURCE\_PLLSAI

RCC\_SAIACLKSOURCE\_PLLI2S

RCC\_SAIACLKSOURCE\_EXT

*RCC SAI BlockB Clock Source*

RCC\_SAIBCLKSOURCE\_PLLSAI

RCC\_SAIBCLKSOURCE\_PLLI2S

RCC\_SAIBCLKSOURCE\_EXT

*RCC SDIO Clock Source*

RCC\_SDIOLCKSOURCE\_CLK48

RCC\_SDIOLCKSOURCE\_SYSCLK

*RCC TIM PRescaler Selection*

RCC\_TIMPRES\_DESACTIVATED

RCC\_TIMPRES\_ACTIVATED

## 56 HAL RNG Generic Driver

### 56.1 RNG Firmware driver registers structures

#### 56.1.1 RNG\_HandleTypeDef

*RNG\_HandleTypeDef* is defined in the `stm32f4xx_hal_rng.h`

##### Data Fields

- *RNG\_TypeDef \* Instance*
- *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
- *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

### 56.2 RNG Firmware driver API description

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

#### 56.2.1 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.

#### 56.2.2 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.

### 56.2.3 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\(\)\*](#)

### 56.2.4 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\\_GetRandomNumber\(\)\*](#)
- [\*HAL\\_RNG\\_GetRandomNumber\\_IT\(\)\*](#)
- [\*HAL\\_RNG\\_IRQHandler\(\)\*](#)
- [\*HAL\\_RNG\\_ReadLastRandomNumber\(\)\*](#)
- [\*HAL\\_RNG\\_ReadyDataCallback\(\)\*](#)
- [\*HAL\\_RNG\\_ErrorCallback\(\)\*](#)

### 56.2.5 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\(\)\*](#)

## 56.2.6 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\_GetRandomNumber

#### Function name

`uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)`

#### Function description

Returns generated random number in polling mode (Obsolete) Use HAL\_RNG\_GenerateRandomNumber() API instead.

#### Parameters

- **hrng**: pointer to a RNG\_HandleTypeDef structure that contains the configuration information for RNG.

#### Return values

- **Random**: value

### HAL\_RNG\_GetRandomNumber\_IT

#### Function name

`uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)`

#### Function description

Returns a 32-bit random number with interrupt enabled (Obsolete), Use HAL\_RNG\_GenerateRandomNumber\_IT() API instead.

#### Parameters

- **hrng**: pointer to a RNG\_HandleTypeDef structure that contains the configuration information for RNG.

#### Return values

- **32-bit**: random number

### 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

- Each time the random number data is read the 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**:

### 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

## 56.3 RNG Firmware driver defines

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

### 56.3.1 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 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`.

## 57 HAL RTC Generic Driver

### 57.1 RTC Firmware driver registers structures

#### 57.1.1 RTC\_InitTypeDef

*RTC\_InitTypeDef* is defined in the `stm32f4xx_hal_rtc.h`

##### Data Fields

- *uint32\_t HourFormat*
- *uint32\_t AsynchPrediv*
- *uint32\_t SynchPrediv*
- *uint32\_t OutPut*
- *uint32\_t OutPutPolarity*
- *uint32\_t OutPutType*

##### 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 = 0x7FFFU`
- *uint32\_t RTC\_InitTypeDef::OutPut*  
Specifies which signal will be routed to the RTC output. This parameter can be a value of [RTC\\_Output\\_selection\\_Definitions](#)
- *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](#)

#### 57.1.2 RTC\_TimeTypeDef

*RTC\_TimeTypeDef* is defined in the `stm32f4xx_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 DayLight Save Operation. This parameter can be a value of [RTC\\_DayLightSaving\\_Definitions](#)
- ***uint32\_t RTC\_TimeTypeDef::StoreOperation***  
Specifies RTC\_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of [RTC\\_StoreOperation\\_Definitions](#)

### 57.1.3 RTC\_DateTypeDef

**RTC\_DateTypeDef** is defined in the stm32f4xx\_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

### 57.1.4 RTC\_AlarmTypeDef

**RTC\_AlarmTypeDef** is defined in the stm32f4xx\_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](#)

### 57.1.5 RTC\_HandleTypeDef

*RTC\_HandleTypeDef* is defined in the `stm32f4xx_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***  
Register 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

## 57.2 RTC Firmware driver API description

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

### 57.2.1 Backup Domain Operating Condition

The real-time clock (RTC), the RTC backup registers, and the backup SRAM (BKP SRAM) can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers, backup SRAM, 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.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

1. The RTC
2. The LSE oscillator
3. The backup SRAM when the low power backup regulator is enabled
4. PC13 to PC15 I/Os, plus PI8 I/O (when available)

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins are available:

1. PC14 and PC15 can be used as either GPIO or LSE pins
2. PC13 can be used as a GPIO or as the RTC\_AF1 pin
3. PI8 can be used as a GPIO or as the RTC\_AF2 pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

1. PC14 and PC15 can be used as LSE pins only

2. PC13 can be used as the RTC\_AF1 pin
3. PI8 can be used as the RTC\_AF2 pin

### 57.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC\_BDCR register to their reset values. The BKPSRAM is not affected by this reset. The only way to reset the BKPSRAM is through the Flash interface by requesting a protection level change from 1 to 0.

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.

### 57.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) 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.

### 57.2.4 How to use this 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.

### 57.2.5 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 wake-up, 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-wake-up mode), by using the RTC alarm or the RTC wake-up events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wake-up from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

#### Callback registration

The compilation define `USE_HAL_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.
- 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.
- 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 WakeUpTimerEventCallback()`. 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.

### 57.2.6 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 13-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 wake-up 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\(\)](#)

### 57.2.7 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\(\)](#)

### 57.2.8 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\(\)](#)

### 57.2.9 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\(\)](#)

### 57.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- [HAL\\_RTC\\_GetState\(\)](#)

### 57.2.11 Detailed description of functions

#### HAL\_RTC\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_RTC\_Init (RTC\_HandleTypeDef \* hrtc)**

##### Function description

Initializes the RTC peripheral.

##### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **HAL:** status

**HAL\_RTC\_DeInit**

**Function name**

**HAL\_StatusTypeDef HAL\_RTC\_DeInit (RTC\_HandleTypeDef \* hrtc)**

**Function description**

DeInitializes the RTC peripheral.

**Parameters**

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **HAL:** status

**Notes**

- This function doesn't reset the RTC Backup Data registers.

**HAL\_RTC\_MspInit**

**Function name**

**void HAL\_RTC\_MspInit (RTC\_HandleTypeDef \* hrtc)**

**Function description**

Initializes the RTC MSP.

**Parameters**

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **None:**

**HAL\_RTC\_MspDeInit**

**Function name**

**void HAL\_RTC\_MspDeInit (RTC\_HandleTypeDef \* hrtc)**

**Function description**

DeInitializes the RTC MSP.

**Parameters**

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **None:**

**HAL\_RTC\_SetTime**

**Function name**

**HAL\_StatusTypeDef HAL\_RTC\_SetTime (RTC\_HandleTypeDef \* hrtc, RTC\_TimeTypeDef \* sTime, uint32\_t Format)**

**Function description**

Sets RTC current time.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Gets RTC current time.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

### 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:  $\text{Second fraction ratio} * \text{time\_unit} = [(\text{SecondFraction} - \text{SubSeconds}) / (\text{SecondFraction} + 1)] * \text{time\_unit}$  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.

### HAL\_RTC\_SetDate

#### Function name

**HAL\_StatusTypeDef HAL\_RTC\_SetDate (RTC\_HandleTypeDef \* hrtc, RTC\_DateTypeDef \* sDate, uint32\_t Format)**

#### Function description

Sets RTC current date.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Gets RTC current date.

### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Sets the specified RTC Alarm.

### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Sets the specified RTC Alarm with Interrupt.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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\_DeactivateAlarm

### Function name

**HAL\_StatusTypeDef HAL\_RTC\_DeactivateAlarm (RTC\_HandleTypeDef \* hrtc, uint32\_t Alarm)**

### Function description

Deactivate the specified RTC Alarm.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Gets the RTC Alarm value and masks.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

This function handles Alarm interrupt request.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### Return values

- **None**:

**HAL\_RTC\_PollForAlarmAEvent**

### Function name

**HAL\_StatusTypeDef HAL\_RTC\_PollForAlarmAEvent (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

### Function description

This function handles AlarmA Polling request.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### Return values

- **None**:

**HAL\_RTC\_WaitForSynchro**

### Function name

**HAL\_StatusTypeDef HAL\_RTC\_WaitForSynchro (RTC\_HandleTypeDef \* hrtc)**

### Function description

Waits until the RTC Time and Date registers (RTC\_TR and RTC\_DR) are synchronized with RTC APB clock.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### 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 wake-up 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

Returns the RTC state.

#### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

#### Return values

- **HAL**: state

### RTC\_EnterInitMode

#### Function name

**HAL\_StatusTypeDef RTC\_EnterInitMode (RTC\_HandleTypeDef \* hrtc)**

#### Function description

Enters the RTC Initialization mode.

#### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

#### Return values

- **HAL**: status

## Notes

- The RTC Initialization mode is write protected, use the `__HAL_RTC_WRITEPROTECTION_DISABLE()` before calling this function.

### RTC\_ByteToBcd2

#### Function name

**uint8\_t RTC\_ByteToBcd2 (uint8\_t Value)**

#### Function description

Converts 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

Converts from 2 digit BCD to Binary.

### Parameters

- **Value:** BCD value to be converted

### Return values

- **Converted:** word

## 57.3 RTC Firmware driver defines

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

### 57.3.1 RTC

RTC

*RTC Alarm Date WeekDay Definitions*

RTC\_ALARMDATEWEEKDAYSEL\_DATE

RTC\_ALARMDATEWEEKDAYSEL\_WEEKDAY

*RTC Alarm Mask Definitions*

RTC\_ALARM\_MASK\_NONE

RTC\_ALARM\_MASK\_DATEWEEKDAY

RTC\_ALARM\_MASK\_HOURS

RTC\_ALARM\_MASK\_MINUTES

RTC\_ALARM\_MASK\_SECONDS

RTC\_ALARM\_MASK\_ALL

*RTC Alarms Definitions*

RTC\_ALARM\_A

RTC\_ALARM\_B

*RTC Alarm Sub Seconds Masks Definitions*

RTC\_ALARMSSUBSECONDMASK\_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC\_ALARMSSUBSECONDMASK\_SS14\_1

SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC\_ALARMSSUBSECONDMASK\_SS14\_2

SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_3

SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_4

SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_5

SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared



#### RTC\_ALARMSSUBSECONDMASK\_SS14\_6

SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_7

SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_8

SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_9

SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_10

SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_11

SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_12

SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14\_13

SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_SS14

SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared

#### RTC\_ALARMSSUBSECONDMASK\_NONE

SS[14:0] are compared and must match to activate alarm.

#### **RTC AM PM Definitions**

#### RTC\_HOURFORMAT12\_AM

#### RTC\_HOURFORMAT12\_PM

#### **RTC DayLight Saving 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__`: specifies the 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\_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 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 Alarm's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag 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 Alarm's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_ALRAF`
  - `RTC_FLAG_ALRBF`

**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_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_GET_FLAG`

**Description:**

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

#### `__HAL_RTC_ALARM_EXTI_CLEAR_FLAG`

**Description:**

- Clear the RTC Alarm associated Exti line flag.

**Return value:**

- None.

## `__HAL_RTC_ALARM_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on RTC Alarm associated Exti line.

**Return value:**

- None.

***RTC Flags Definitions***

`RTC_FLAG_RECALPF`

`RTC_FLAG_TAMP2F`

`RTC_FLAG_TAMP1F`

`RTC_FLAG_TSOVF`

`RTC_FLAG_TSF`

`RTC_FLAG_WUTF`

`RTC_FLAG_ALRBF`

`RTC_FLAG_ALRAF`

`RTC_FLAG_INITF`

`RTC_FLAG_RSOF`

`RTC_FLAG_INITS`

`RTC_FLAG_SHPF`

`RTC_FLAG_WUTWF`

`RTC_FLAG_ALRBWF`

`RTC_FLAG_ALRAWF`

***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`

`RTC_IT_WUT`

`RTC_IT_ALRB`

`RTC_IT_ALRA`

RTC\_IT\_TAMP

RTC\_IT\_TAMP1

RTC\_IT\_TAMP2

*RTC Private macros to check input parameters*

IS\_RTC\_HOUR\_FORMAT

IS\_RTC\_OUTPUT

IS\_RTC\_OUTPUT\_POL

IS\_RTC\_OUTPUT\_TYPE

IS\_RTC\_HOUR12

IS\_RTC\_HOUR24

IS\_RTC\_ASYNC\_PREDIV

IS\_RTC\_SYNC\_PREDIV

IS\_RTC\_MINUTES

IS\_RTC\_SECONDS

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

***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 Polarity Definitions***

RTC\_OUTPUT\_POLARITY\_HIGH

RTC\_OUTPUT\_POLARITY\_LOW

***RTC Output Selection Definitions***

RTC\_OUTPUT\_DISABLE

RTC\_OUTPUT\_ALARMA

RTC\_OUTPUT\_ALARMB

RTC\_OUTPUT\_WAKEUP

***RTC Output Type ALARM OUT***

RTC\_OUTPUT\_TYPE\_OPENDRAIN

RTC\_OUTPUT\_TYPE\_PUSHPULL

***RTC Store Operation 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

## 58 HAL RTC Extension Driver

### 58.1 RTCEX Firmware driver registers structures

#### 58.1.1 RTC\_TamperTypeDef

*RTC\_TamperTypeDef* is defined in the `stm32f4xx_hal_rtc_ex.h`

##### Data Fields

- *uint32\_t Tamper*
- *uint32\_t PinSelection*
- *uint32\_t Trigger*
- *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\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::PinSelection*  
Specifies the Tamper Pin. This parameter can be a value of [RTCEX\\_Tamper\\_Pins\\_Selection](#)
- *uint32\_t RTC\_TamperTypeDef::Trigger*  
Specifies the Tamper Trigger. This parameter can be a value of [RTCEX\\_Tamper\\_Trigger\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::Filter*  
Specifies the RTC Filter Tamper. This parameter can be a value of [RTCEX\\_Tamper\\_Filter\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::SamplingFrequency*  
Specifies the sampling frequency. This parameter can be a value of [RTCEX\\_Tamper\\_Sampling\\_Frequencies\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::PrechargeDuration*  
Specifies the Precharge Duration . This parameter can be a value of [RTCEX\\_Tamper\\_Pin\\_Precharge\\_Duration\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::TamperPullUp*  
Specifies the Tamper PullUp . This parameter can be a value of [RTCEX\\_Tamper\\_Pull\\_UP\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::TimeStampOnTamperDetection*  
Specifies the TimeStampOnTamperDetection. This parameter can be a value of [RTCEX\\_Tamper\\_TimeStampOnTamperDetection\\_Definitions](#)

### 58.2 RTCEX Firmware driver API description

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

#### 58.2.1 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 Wake-up configuration

- To configure the RTC Wake-up Clock source and Counter use the `HAL_RTCEX_SetWakeUpTimer()` function. You can also configure the RTC Wake-up timer in interrupt mode using the `HAL_RTCEX_SetWakeUpTimer_IT()` function.
- To read the RTC Wake-up Counter register, use the `HAL_RTCEX_GetWakeUpTimer()` function.

### TimeStamp configuration

- Configure the RTC\_AFx trigger and 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.
- The TIMESTAMP alternate function can be mapped either to RTC\_AF1 (PC13) or RTC\_AF2 (PI8 or PA0 only for STM32F446xx devices) depending on the value of TSINSEL bit in RTC\_TAFPCR register. The corresponding pin is also selected by HAL\_RTCEX\_SetTimeStamp() or HAL\_RTCEX\_SetTimeStamp\_IT() 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, precharge or discharge and Pull-UP using the HAL\_RTCEX\_SetTamper() function. You can configure RTC Tamper in interrupt mode using HAL\_RTCEX\_SetTamper\_IT() function.
- The TAMPER1 alternate function can be mapped either to RTC\_AF1 (PC13) or RTC\_AF2 (PI8 or PA0 only for STM32F446xx devices) depending on the value of TAMP1INSEL bit in RTC\_TAFPCR register. The corresponding pin is also selected by HAL\_RTCEX\_SetTamper() or HAL\_RTCEX\_SetTamper\_IT() function.

### 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.

## 58.2.2 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\\_GetTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEX\\_SetTamper\(\)\*](#)
- [\*HAL\\_RTCEX\\_SetTamper\\_IT\(\)\*](#)
- [\*HAL\\_RTCEX\\_DeactivateTamper\(\)\*](#)
- [\*HAL\\_RTCEX\\_TamperTimeStampIRQHandler\(\)\*](#)
- [\*HAL\\_RTCEX\\_TimeStampEventCallback\(\)\*](#)
- [\*HAL\\_RTCEX\\_Tamper1EventCallback\(\)\*](#)
- [\*HAL\\_RTCEX\\_Tamper2EventCallback\(\)\*](#)
- [\*HAL\\_RTCEX\\_PollForTimeStampEvent\(\)\*](#)
- [\*HAL\\_RTCEX\\_PollForTamper1Event\(\)\*](#)
- [\*HAL\\_RTCEX\\_PollForTamper2Event\(\)\*](#)

## 58.2.3 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\(\)\*](#)

### 58.2.4 Extension 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\\_BKUPWrite\(\)\*](#)
- [\*HAL\\_RTCEx\\_BKUPRead\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetCoarseCalib\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateCoarseCalib\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetSmoothCalib\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetSynchroShift\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetCalibrationOutPut\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateCalibrationOutPut\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetRefClock\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateRefClock\(\)\*](#)
- [\*HAL\\_RTCEx\\_EnableBypassShadow\(\)\*](#)
- [\*HAL\\_RTCEx\\_DisableBypassShadow\(\)\*](#)

### 58.2.5 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\(\)\*](#)

### 58.2.6 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

Sets TimeStamp.

### Parameters

- **hrtc**: pointer to a `RTC_HandleTypeDef` structure that contains the configuration information for RTC.
- **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.
  - `RTC_TIMESTAMPPIN_POS1`: PI8/PA0 is selected as RTC TimeStamp Pin. (not applicable in the case of STM32F412xx, STM32F413xx and STM32F423xx devices) (PI8 for all STM32 devices except for STM32F446xx devices the PA0 is used)
  - `RTC_TIMESTAMPPIN_PA0`: PA0 is selected as RTC TimeStamp Pin only for STM32F446xx devices

### 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

Sets TimeStamp with Interrupt.

### Parameters

- **hrtc**: pointer to a `RTC_HandleTypeDef` structure that contains the configuration information for RTC.
- **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.
  - `RTC_TIMESTAMPPIN_PI8`: PI8 is selected as RTC TimeStamp Pin. (not applicable in the case of STM32F446xx, STM32F412xx, STM32F413xx and STM32F423xx devices)
  - `RTC_TIMESTAMPPIN_PA0`: PA0 is selected as RTC TimeStamp Pin only for STM32F446xx devices

### 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

Deactivates TimeStamp.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### 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

Gets the RTC TimeStamp value.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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\_SetTamper

### Function name

**HAL\_StatusTypeDef HAL\_RTCEx\_SetTamper (RTC\_HandleTypeDef \* hrtc, RTC\_TamperTypeDef \* sTamper)**

### Function description

Sets Tamper.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **sTamper**: Pointer to Tamper Structure.

### Return values

- **HAL**: status

### Notes

- By calling this API we disable the tamper interrupt for all tampers.

### HAL\_RTCEx\_SetTamper\_IT

### Function name

**HAL\_StatusTypeDef HAL\_RTCEx\_SetTamper\_IT (RTC\_HandleTypeDef \* hrtc, RTC\_TamperTypeDef \* sTamper)**

### Function description

Sets Tamper with interrupt.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **sTamper**: Pointer to RTC Tamper.

**Return values**

- **HAL**: status

**Notes**

- By calling this API we force the tamper interrupt for all tampers.

**HAL\_RTCEx\_DeactivateTamper**
**Function name**

**HAL\_StatusTypeDef HAL\_RTCEx\_DeactivateTamper (RTC\_HandleTypeDef \* hrtc, uint32\_t Tamper)**

**Function description**

Deactivates Tamper.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Tamper**: Selected tamper pin. This parameter can be RTC\_Tamper\_1 and/or RTC\_TAMPER\_2.

**Return values**

- **HAL**: status

**HAL\_RTCEx\_TamperTimeStampIRQHandler**
**Function name**

**void HAL\_RTCEx\_TamperTimeStampIRQHandler (RTC\_HandleTypeDef \* hrtc)**

**Function description**

This function handles TimeStamp interrupt request.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **None**:

**HAL\_RTCEx\_Tamper1EventCallback**
**Function name**

**void HAL\_RTCEx\_Tamper1EventCallback (RTC\_HandleTypeDef \* hrtc)**

**Function description**

Tamper 1 callback.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **None**:

**HAL\_RTCEx\_Tamper2EventCallback**
**Function name**

**void HAL\_RTCEx\_Tamper2EventCallback (RTC\_HandleTypeDef \* hrtc)**

**Function description**

Tamper 2 callback.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **None**:

**HAL\_RTCEx\_TimeStampEventCallback**

**Function name**

**void HAL\_RTCEx\_TimeStampEventCallback (RTC\_HandleTypeDef \* hrtc)**

**Function description**

TimeStamp callback.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **None**:

**HAL\_RTCEx\_PollForTimeStampEvent**

**Function name**

**HAL\_StatusTypeDef HAL\_RTCEx\_PollForTimeStampEvent (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

**Function description**

This function handles TimeStamp polling request.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout**: Timeout duration

**Return values**

- **HAL**: status

**HAL\_RTCEx\_PollForTamper1Event**

**Function name**

**HAL\_StatusTypeDef HAL\_RTCEx\_PollForTamper1Event (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

**Function description**

This function handles Tamper1 Polling.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout**: Timeout duration

**Return values**

- **HAL**: status

**HAL\_RTCEx\_PollForTamper2Event**

**Function name**

**HAL\_StatusTypeDef HAL\_RTCEx\_PollForTamper2Event (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**



### Function description

This function handles Tamper2 Polling.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

### HAL\_RTCEx\_SetWakeUpTimer

### Function name

**HAL\_StatusTypeDef HAL\_RTCEx\_SetWakeUpTimer (RTC\_HandleTypeDef \* hrtc, uint32\_t WakeUpCounter, uint32\_t WakeUpClock)**

### Function description

Sets wake up timer.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Sets wake up timer with interrupt.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **WakeUpCounter**: Wake up counter
- **WakeUpClock**: Wake up clock

### Return values

- **HAL**: status

### HAL\_RTCEx\_DeactivateWakeUpTimer

### Function name

**uint32\_t HAL\_RTCEx\_DeactivateWakeUpTimer (RTC\_HandleTypeDef \* hrtc)**

### Function description

Deactivates wake up timer counter.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### Return values

- **HAL**: status

### HAL\_RTCEx\_GetWakeUpTimer

#### Function name

`uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)`

#### Function description

Gets wake up timer counter.

#### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

#### Return values

- **Counter:** value

### HAL\_RTCEx\_WakeUpTimerIRQHandler

#### Function name

`void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)`

#### Function description

This function handles Wake Up Timer interrupt request.

#### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

#### Return values

- **None:**

#### Notes

- Unlike alarm interrupt line (shared by AlarmA and AlarmB) and tamper interrupt line (shared by timestamp and tampers) wakeup timer interrupt line is exclusive to the wakeup timer. There is no need in this case to check on the interrupt enable status via `__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE()`.

### HAL\_RTCEx\_WakeUpTimerEventCallback

#### Function name

`void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)`

#### Function description

Wake Up Timer callback.

#### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

#### Return values

- **None:**

### HAL\_RTCEx\_PollForWakeUpTimerEvent

#### Function name

`HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)`

#### Function description

This function handles Wake Up Timer Polling.

### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout:** Timeout duration

### Return values

- **HAL:** status

### HAL\_RTCEx\_BKUPWrite

#### Function name

```
void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)
```

#### Function description

Writes a data in a specified RTC Backup data register.

### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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 RTC 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 RTC Backup data Register.

### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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.

### Return values

- **Read:** value

### HAL\_RTCEx\_SetCoarseCalib

#### Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetCoarseCalib (RTC_HandleTypeDef * hrtc, uint32_t CalibSign, uint32_t Value)
```

#### Function description

Sets the Coarse calibration parameters.

### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **CalibSign:** Specifies the sign of the coarse calibration value. This parameter can be one of the following values :
  - RTC\_CALIBSIGN\_POSITIVE: The value sign is positive
  - RTC\_CALIBSIGN\_NEGATIVE: The value sign is negative
- **Value:** value of coarse calibration expressed in ppm (coded on 5 bits).

### Return values

- **HAL:** status

### Notes

- This Calibration value should be between 0 and 63 when using negative sign with a 2-ppm step.
- This Calibration value should be between 0 and 126 when using positive sign with a 4-ppm step.

### HAL\_RTCEX\_DeactivateCoarseCalib

#### Function name

**HAL\_StatusTypeDef HAL\_RTCEX\_DeactivateCoarseCalib (RTC\_HandleTypeDef \* hrtc)**

#### Function description

Deactivates the Coarse calibration parameters.

#### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### 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

Sets the Smooth calibration parameters.

#### Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter 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[80] bits. This parameter can be 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

Configures the Synchronization Shift Control Settings.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Configures the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).

#### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Deactivates the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).

#### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

#### Return values

- **HAL**: status

### HAL\_RTCEX\_SetRefClock

#### Function name

**HAL\_StatusTypeDef HAL\_RTCEX\_SetRefClock (RTC\_HandleTypeDef \* hrtc)**

#### Function description

Enables the RTC reference clock detection.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**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**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**Return values**

- **HAL**: status

**HAL\_RTCEx\_EnableBypassShadow**
**Function name**

**HAL\_StatusTypeDef HAL\_RTCEx\_EnableBypassShadow (RTC\_HandleTypeDef \* hrtc)**

**Function description**

Enables the Bypass Shadow feature.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**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**

Disables the Bypass Shadow feature.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

**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**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.

### Return values

- **None**:

**HAL\_RTCEX\_PollForAlarmBEvent**

### Function name

**HAL\_StatusTypeDef HAL\_RTCEX\_PollForAlarmBEvent (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

### Function description

This function handles AlarmB Polling request.

### Parameters

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

## 58.3 RTCEX Firmware driver defines

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

### 58.3.1 RTCEX

RTCEX

*RTC Add 1 Second Parameter Definitions*

**RTC\_SHIFTADD1S\_RESET**

**RTC\_SHIFTADD1S\_SET**

*RTC Backup Registers Definitions*

**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 Calibration*

#### **\_\_HAL\_RTC\_COARSE\_CALIB\_ENABLE**

**Description:**

- Enable the Coarse calibration process.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

#### **\_\_HAL\_RTC\_COARSE\_CALIB\_DISABLE**

**Description:**

- Disable the Coarse calibration process.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

#### **\_\_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 operation's 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

***RTC Calib Output Selection Definitions***

**RTC\_CALIBOUTPUT\_512HZ**

**RTC\_CALIBOUTPUT\_1HZ**

***RTC Digital Calib Definitions***

**RTC\_CALIBSIGN\_POSITIVE**

**RTC\_CALIBSIGN\_NEGATIVE**

***Private macros to check input parameters***

**IS\_RTC\_BKP**

**IS\_TIMESTAMP\_EDGE**

**IS\_RTC\_TAMPER**

IS\_RTC\_TAMPER\_PIN

IS\_RTC\_TIMESTAMP\_PIN

IS\_RTC\_TAMPER\_TRIGGER

IS\_RTC\_TAMPER\_FILTER

IS\_RTC\_TAMPER\_SAMPLING\_FREQ

IS\_RTC\_TAMPER\_PRECHARGE\_DURATION

IS\_RTC\_TAMPER\_TIMESTAMPONTAMPER\_DETECTION

IS\_RTC\_TAMPER\_PULLUP\_STATE

IS\_RTC\_WAKEUP\_CLOCK

IS\_RTC\_WAKEUP\_COUNTER

IS\_RTC\_CALIB\_SIGN

IS\_RTC\_CALIB\_VALUE

IS\_RTC\_SMOOTH\_CALIB\_PERIOD

IS\_RTC\_SMOOTH\_CALIB\_PLUS

IS\_RTC\_SMOOTH\_CALIB\_MINUS

IS\_RTC\_SHIFT\_ADD1S

IS\_RTC\_SHIFT\_SUBFS

IS\_RTC\_CALIB\_OUTPUT

***RTC Smooth Calib Period Definitions***

**RTC\_SMOOTHCALIB\_PERIOD\_32SEC**

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else  $2 \times \exp(20)$  RTCCLK seconds

**RTC\_SMOOTHCALIB\_PERIOD\_16SEC**

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else  $2 \times \exp(19)$  RTCCLK seconds

**RTC\_SMOOTHCALIB\_PERIOD\_8SEC**

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else  $2 \times \exp(18)$  RTCCLK seconds

***RTC Smooth Calib Plus Pulses Definitions***

**RTC\_SMOOTHCALIB\_PLUSPULSES\_SET**

The number of RTCCLK pulses added during a X -second window =  $Y - \text{CALM}[8:0]$  with Y = 512, 256, 128 when X = 32, 16, 8

**RTC\_SMOOTHCALIB\_PLUSPULSES\_RESET**

The number of RTCCLK pulses substituted during a 32-second window =  $\text{CALM}[8:0]$

***RTC Tamper***

### **\_\_HAL\_RTC\_TAMPER1\_ENABLE**

**Description:**

- Enable the RTC Tamper1 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER1\_DISABLE**

**Description:**

- Disable the RTC Tamper1 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER2\_ENABLE**

**Description:**

- Enable the RTC Tamper2 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER2\_DISABLE**

**Description:**

- Disable the RTC Tamper2 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER\_GET\_IT**

**Description:**

- Check whether the specified RTC 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_TAMP1`
  - `RTC_IT_TAMP2`

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER\_GET\_IT\_SOURCE**

**Description:**

- Check whether the specified RTC 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`: Tamper interrupt

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER\_GET\_FLAG**

**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_TAMP1F`
  - `RTC_FLAG_TAMP2F`

**Return value:**

- None

### **\_\_HAL\_RTC\_TAMPER\_CLEAR\_FLAG**

**Description:**

- Clear the RTC Tamper's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag to clear. This parameter can be:
  - `RTC_FLAG_TAMP1F`
  - `RTC_FLAG_TAMP2F`

**Return value:**

- None

***RTC Tamper Filter Definitions***

#### **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.

***RTC Tamper Pins Definitions***

#### **RTC\_TAMPER\_1**

#### **RTC\_TAMPER\_2**

***RTC tamper Pins Selection***

## RTC\_TAMPERPIN\_DEFAULT

## RTC\_TAMPERPIN\_POS1

### *RTC Tamper Pin Precharge Duration Definitions*

#### 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

### *RTC Tamper Pull Up Definitions*

#### RTC\_TAMPER\_PULLUP\_ENABLE

TimeStamp on Tamper Detection event saved

#### RTC\_TAMPER\_PULLUP\_DISABLE

TimeStamp on Tamper Detection event is not saved

### *RTC Tamper Sampling Frequencies Definitions*

#### 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 =  $RTCCLK / 256$

### *EXTI RTC Tamper Timestamp EXTI*

#### \_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_ENABLE\_IT

##### **Description:**

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

##### **Return value:**

- None

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT`

**Description:**

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT`

**Description:**

- Enable event on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT`

**Description:**

- Disable event on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE`

**Description:**

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE`

**Description:**

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE`

**Description:**

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE`

**Description:**

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

#### `__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE`

**Description:**

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

### **\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_DISABLE\_RISING\_FALLING\_EDGE**

**Description:**

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

### **\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_GET\_FLAG**

**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

### **\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_CLEAR\_FLAG**

**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

**Return value:**

- None.

### **\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_GENERATE\_SWIT**

**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

***RTC Tamper TimeStamp On Tamper Detection Definitions***

### **RTC\_TIMESTAMPONTAMPERDETECTION\_ENABLE**

TimeStamp on Tamper Detection event saved

### **RTC\_TIMESTAMPONTAMPERDETECTION\_DISABLE**

TimeStamp on Tamper Detection event is not saved

***RTC Tamper Triggers Definitions***

### **RTC\_TAMPERTRIGGER\_RISINGEDGE**

### **RTC\_TAMPERTRIGGER\_FALLINGEDGE**

### **RTC\_TAMPERTRIGGER\_LOWLEVEL**

### **RTC\_TAMPERTRIGGER\_HIGHLEVEL**

***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 sources to be enabled or disabled. 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 sources to be enabled or 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 TimeStamp's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC TimeStamp flag to check. This parameter can be:
  - `RTC_FLAG_TSF`
  - `RTC_FLAG_TSOVF`

**Return value:**

- None

### **\_\_HAL\_RTC\_TIMESTAMP\_CLEAR\_FLAG**

**Description:**

- Clear the RTC Time Stamp's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_TSF`

**Return value:**

- None

***RTC TimeStamp Pins Selection***

### **RTC\_TIMESTAMPPIN\_DEFAULT**

### **RTC\_TIMESTAMPPIN\_POS1**

***RTC TimeStamp Edges Definitions***

### **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 Wake-up 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 or disabled. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer A 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 enabled or disabled. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer A 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 A 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 WakeUpTimer's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC WakeUpTimer Flag to check. This parameter can be:
  - `RTC_FLAG_WUTF`
  - `RTC_FLAG_WUTWF`

**Return value:**

- None

### **\_\_HAL\_RTC\_WAKEUPTIMER\_CLEAR\_FLAG**

**Description:**

- Clear the RTC Wake Up timer's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_WUTF`

**Return value:**

- None

### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_IT**

**Description:**

- Enable interrupt on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None

### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_IT**

**Description:**

- Disable interrupt on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None

### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_EVENT**

**Description:**

- Enable event on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_EVENT**

**Description:**

- Disable event on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_FALLING\_EDGE**

**Description:**

- Enable falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_FALLING\_EDGE**

**Description:**

- Disable falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_RISING\_EDGE**

**Description:**

- Enable rising edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_RISING\_EDGE**

**Description:**

- Disable rising edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE**

**Description:**

- Enable rising & falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_RISING\_FALLING\_EDGE**

**Description:**

- Disable rising & falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_GET\_FLAG**

**Description:**

- Check whether the RTC Wake-up Timer associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_CLEAR\_FLAG**

**Description:**

- Clear the RTC Wake-up Timer associated Exti line flag.

**Return value:**

- None.

#### **\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_GENERATE\_SWIT**

**Description:**

- Generate a Software interrupt on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

***RTC Wake-up 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

## 59 HAL SAI Generic Driver

### 59.1 SAI Firmware driver registers structures

#### 59.1.1 SAI\_InitTypeDef

*SAI\_InitTypeDef* is defined in the `stm32f4xx_hal_sai.h`

##### Data Fields

- *uint32\_t AudioMode*
- *uint32\_t Synchro*
- *uint32\_t SynchroExt*
- *uint32\_t OutputDrive*
- *uint32\_t NoDivider*
- *uint32\_t FIFOThreshold*
- *uint32\_t ClockSource*
- *uint32\_t AudioFrequency*
- *uint32\_t Mckdiv*
- *uint32\_t MonoStereoMode*
- *uint32\_t CompandingMode*
- *uint32\_t TriState*
- *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::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 without constraint since the input clock of the audio block should be equal to the bit clock. There is no MCLK\_x clock which can be output.
- *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::ClockSource*  
Specifies the SAI Block x Clock source. This parameter is not used for STM32F446xx devices.

- ***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 = 15.  
**Note:**
  - This parameter is used only if AudioFrequency is set to SAI\_AUDIO\_FREQUENCY\_MCKDIV otherwise it is internally computed.
- ***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\\_TRISate\\_Management](#)
- ***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](#)

### 59.1.2

#### SAI\_FramelnitTypeDef

**SAI\_FramelnitTypeDef** is defined in the `stm32f4xx_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\_FramelnitTypeDef::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\_FramelnitTypeDef::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\_FramelnitTypeDef::FSDefinition***  
Specifies the Frame synchronization definition. This parameter can be a value of [SAI\\_Block\\_FS\\_Definition](#)
- ***uint32\_t SAI\_FramelnitTypeDef::FSPolarity***  
Specifies the Frame synchronization Polarity. This parameter can be a value of [SAI\\_Block\\_FS\\_Polarity](#)
- ***uint32\_t SAI\_FramelnitTypeDef::FSOffset***  
Specifies the Frame synchronization Offset. This parameter can be a value of [SAI\\_Block\\_FS\\_Offset](#)

### 59.1.3 SAI\_SlotInitTypeDef

**SAI\_SlotInitTypeDef** is defined in the stm32f4xx\_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](#)

### 59.1.4 \_\_SAI\_HandleTypeDef

**\_\_SAI\_HandleTypeDef** is defined in the stm32f4xx\_hal\_sai.h

#### Data Fields

- *SAI\_Block\_TypeDef \* Instance*
- *SAI\_InitTypeDef Init*
- *SAI\_FrameInitTypeDef FrameInit*
- *SAI\_SlotInitTypeDef SlotInit*
- *uint8\_t \* pBuffer*
- *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::pBuffer***  
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

## 59.2 SAI Firmware driver API description

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

### 59.2.1 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`, `Framelnit` and `Slotlnit` 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:* SAI Clock Source configuration is managed differently depending on the selected STM32F4 devices :

- For STM32F446xx devices, the configuration is managed through `RCCEX_PeriphCLKConfig()` function in the HAL RCC drivers
- For STM32F439xx/STM32F437xx/STM32F429xx/STM32F427xx devices, the configuration is managed within HAL SAI drivers through `HAL_SAI_Init()` function using `ClockSource` field of `SAI_InitTypeDef` structure.

**Note:** *Make sure that either:*

- *I2S PLL is configured or*
- *SAI PLL is configured or*
- *External clock source is configured after setting correctly the define constant EXTERNAL\_CLOCK\_VALUE in the stm32f4xx\_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.*

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.

## 59.2.2 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
- 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\\_MspltInit\(\)\*](#)
- [\*HAL\\_SAI\\_MspDeInit\(\)\*](#)

### 59.2.3 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\\_DMABasePause\(\)\*](#)
- [\*HAL\\_SAI\\_DMABaseResume\(\)\*](#)
- [\*HAL\\_SAI\\_DMABaseStop\(\)\*](#)
- [\*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()*

#### 59.2.4 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()*

#### 59.2.5 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 Framelnit, Slotlnit 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 \* h sai, 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 \* h sai)**

#### 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 \* h sai)**

#### 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 \* h sai)**

#### 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

## 59.3 SAI Firmware driver defines

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

### 59.3.1

#### 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 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 Clock Source**

SAI\_CLKSOURCE\_PLLSAI

SAI\_CLKSOURCE\_PLLI2S

SAI\_CLKSOURCE\_EXT

SAI\_CLKSOURCE\_NA

No applicable for STM32F446xx

**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 or 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\_MUTEDDET: 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

## \_\_HAL\_SAI\_GET\_IT\_SOURCE

**Description:**

- Check if the specified SAI interrupt source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: specifies the SAI Handle. This parameter can be SAI where x: 1, 2, or 3 to select the SAI peripheral.
- `__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 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 TRISState Management***

### SAI\_OUTPUT\_NOTRELEASED

### SAI\_OUTPUT\_RELEASED

## 60 HAL SAI Extension Driver

### 60.1 SAIEx Firmware driver API description

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

#### 60.1.1 SAI peripheral extension features

Comparing to other previous devices, the SAI interface for STM32F446xx devices contains the following additional features :

- Possibility to be clocked from PLLR

#### 60.1.2 How to use this driver

This driver provides functions to manage several sources to clock SAI

#### 60.1.3 Extension features Functions

This subsection provides a set of functions allowing to manage the possible SAI clock sources.

This section contains the following APIs:

- [SAI\\_BlockSynchroConfig\(\)](#)
- [SAI\\_GetInputClock\(\)](#)

#### 60.1.4 Detailed description of functions

##### SAI\_BlockSynchroConfig

###### Function name

**void SAI\_BlockSynchroConfig (SAI\_HandleTypeDef \* hsai)**

###### Function description

Configure SAI Block synchronization mode.

###### Parameters

- **hsai**: pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

###### Return values

- **SAI**: Clock Input

##### SAI\_GetInputClock

###### Function name

**uint32\_t SAI\_GetInputClock (SAI\_HandleTypeDef \* hsai)**

###### Function description

Get SAI Input Clock based on SAI source clock selection.

###### Parameters

- **hsai**: pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

###### Return values

- **SAI**: Clock Input

### 60.2 SAIEx Firmware driver defines

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

60.2.1 **SAIEx**  
SAIEx

## 61 HAL SD Generic Driver

### 61.1 SD Firmware driver registers structures

#### 61.1.1 HAL\_SD\_CardInfoTypeDef

*HAL\_SD\_CardInfoTypeDef* is defined in the `stm32f4xx_hal_sd.h`

Data Fields

- *uint32\_t CardType*
- *uint32\_t CardVersion*
- *uint32\_t Class*
- *uint32\_t RelCardAdd*
- *uint32\_t BlockNbr*
- *uint32\_t BlockSize*
- *uint32\_t LogBlockNbr*
- *uint32\_t LogBlockSize*

Field Documentation

- *uint32\_t HAL\_SD\_CardInfoTypeDef::CardType*  
Specifies the card Type
- *uint32\_t HAL\_SD\_CardInfoTypeDef::CardVersion*  
Specifies the card version
- *uint32\_t HAL\_SD\_CardInfoTypeDef::Class*  
Specifies the class of the card class
- *uint32\_t HAL\_SD\_CardInfoTypeDef::RelCardAdd*  
Specifies the Relative Card Address
- *uint32\_t HAL\_SD\_CardInfoTypeDef::BlockNbr*  
Specifies the Card Capacity in blocks
- *uint32\_t HAL\_SD\_CardInfoTypeDef::BlockSize*  
Specifies one block size in bytes
- *uint32\_t HAL\_SD\_CardInfoTypeDef::LogBlockNbr*  
Specifies the Card logical Capacity in blocks
- *uint32\_t HAL\_SD\_CardInfoTypeDef::LogBlockSize*  
Specifies logical block size in bytes

#### 61.1.2 SD\_HandleTypeDef

*SD\_HandleTypeDef* is defined in the `stm32f4xx_hal_sd.h`

Data Fields

- *SD\_TypeDef \* Instance*
- *SD\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *uint8\_t \* pTxBuffPtr*
- *uint32\_t TxXferSize*
- *uint8\_t \* pRxBuffPtr*
- *uint32\_t RxXferSize*
- *\_\_IO uint32\_t Context*
- *\_\_IO HAL\_SD\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_SD\_CardInfoTypeDef SdCard*

- ***uint32\_t CSD***
- ***uint32\_t CID***

**Field Documentation**

- ***SD\_TypeDef\* SD\_HandleTypeDef::Instance***  
SD registers base address
- ***SD\_InitTypeDef SD\_HandleTypeDef::Init***  
SD required parameters
- ***HAL\_LockTypeDef SD\_HandleTypeDef::Lock***  
SD locking object
- ***uint8\_t\* SD\_HandleTypeDef::pTxBuffPtr***  
Pointer to SD Tx transfer Buffer
- ***uint32\_t SD\_HandleTypeDef::TxXferSize***  
SD Tx Transfer size
- ***uint8\_t\* SD\_HandleTypeDef::pRxBuffPtr***  
Pointer to SD Rx transfer Buffer
- ***uint32\_t SD\_HandleTypeDef::RxXferSize***  
SD Rx Transfer size
- ***\_\_IO uint32\_t SD\_HandleTypeDef::Context***  
SD transfer context
- ***\_\_IO HAL\_SD\_StateTypeDef SD\_HandleTypeDef::State***  
SD card State
- ***\_\_IO uint32\_t SD\_HandleTypeDef::ErrorCode***  
SD Card Error codes
- ***DMA\_HandleTypeDef\* SD\_HandleTypeDef::hdmatx***  
SD Tx DMA handle parameters
- ***DMA\_HandleTypeDef\* SD\_HandleTypeDef::hdmarx***  
SD Rx DMA handle parameters
- ***HAL\_SD\_CardInfoTypeDef SD\_HandleTypeDef::SdCard***  
SD Card information
- ***uint32\_t SD\_HandleTypeDef::CSD[4]***  
SD card specific data table
- ***uint32\_t SD\_HandleTypeDef::CID[4]***  
SD card identification number table

**61.1.3**
**HAL\_SD\_CardCSDTypeDef**

**HAL\_SD\_CardCSDTypeDef** is defined in the stm32f4xx\_hal\_sd.h

**Data Fields**

- ***\_\_IO uint8\_t CSDStruct***
- ***\_\_IO uint8\_t SysSpecVersion***
- ***\_\_IO uint8\_t Reserved1***
- ***\_\_IO uint8\_t TAAC***
- ***\_\_IO uint8\_t NSAC***
- ***\_\_IO uint8\_t MaxBusClkFrec***
- ***\_\_IO uint16\_t CardComdClasses***
- ***\_\_IO uint8\_t RdBlockLen***
- ***\_\_IO uint8\_t PartBlockRead***
- ***\_\_IO uint8\_t WrBlockMisalign***
- ***\_\_IO uint8\_t RdBlockMisalign***
- ***\_\_IO uint8\_t DSRImpl***
- ***\_\_IO uint8\_t Reserved2***

- `__IO uint32_t DeviceSize`
- `__IO uint8_t MaxRdCurrentVDDMin`
- `__IO uint8_t MaxRdCurrentVDDMax`
- `__IO uint8_t MaxWrCurrentVDDMin`
- `__IO uint8_t MaxWrCurrentVDDMax`
- `__IO uint8_t DeviceSizeMul`
- `__IO uint8_t EraseGrSize`
- `__IO uint8_t EraseGrMul`
- `__IO uint8_t WrProtectGrSize`
- `__IO uint8_t WrProtectGrEnable`
- `__IO uint8_t ManDefIECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGroup`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

#### Field Documentation

- `__IO uint8_t HAL_SD_CardCSDTypeDef::CSDStruct`  
CSD structure
- `__IO uint8_t HAL_SD_CardCSDTypeDef::SysSpecVersion`  
System specification version
- `__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved1`  
Reserved
- `__IO uint8_t HAL_SD_CardCSDTypeDef::TAAC`  
Data read access time 1
- `__IO uint8_t HAL_SD_CardCSDTypeDef::NSAC`  
Data read access time 2 in CLK cycles
- `__IO uint8_t HAL_SD_CardCSDTypeDef::MaxBusClkFrec`  
Max. bus clock frequency
- `__IO uint16_t HAL_SD_CardCSDTypeDef::CardComdClasses`  
Card command classes
- `__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockLen`  
Max. read data block length
- `__IO uint8_t HAL_SD_CardCSDTypeDef::PartBlockRead`  
Partial blocks for read allowed
- `__IO uint8_t HAL_SD_CardCSDTypeDef::WrBlockMisalign`  
Write block misalignment
- `__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockMisalign`  
Read block misalignment
- `__IO uint8_t HAL_SD_CardCSDTypeDef::DSRImpl`  
DSR implemented



- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::Reserved2**  
Reserved
- **\_\_IO uint32\_t HAL\_SD\_CardCSDTypeDef::DeviceSize**  
Device Size
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::MaxRdCurrentVDDMin**  
Max. read current @ VDD min
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::MaxRdCurrentVDDMax**  
Max. read current @ VDD max
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::MaxWrCurrentVDDMin**  
Max. write current @ VDD min
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::MaxWrCurrentVDDMax**  
Max. write current @ VDD max
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::DeviceSizeMul**  
Device size multiplier
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::EraseGrSize**  
Erase group size
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::EraseGrMul**  
Erase group size multiplier
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::WrProtectGrSize**  
Write protect group size
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::WrProtectGrEnable**  
Write protect group enable
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::ManDeflECC**  
Manufacturer default ECC
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::WrSpeedFact**  
Write speed factor
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::MaxWrBlockLen**  
Max. write data block length
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::WriteBlockPaPartial**  
Partial blocks for write allowed
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::Reserved3**  
Reserved
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::ContentProtectAppli**  
Content protection application
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::FileFormatGroup**  
File format group
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::CopyFlag**  
Copy flag (OTP)
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::PermWrProtect**  
Permanent write protection
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::TempWrProtect**  
Temporary write protection
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::FileFormat**  
File format
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::ECC**  
ECC code
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::CSD\_CRC**  
CSD CRC
- **\_\_IO uint8\_t HAL\_SD\_CardCSDTypeDef::Reserved4**  
Always 1

### 61.1.4 HAL\_SD\_CardCIDTypeDef

*HAL\_SD\_CardCIDTypeDef* is defined in the `stm32f4xx_hal_sd.h`

#### Data Fields

- `__IO uint8_t ManufacturerID`
- `__IO uint16_t OEM_AppliID`
- `__IO uint32_t ProdName1`
- `__IO uint8_t ProdName2`
- `__IO uint8_t ProdRev`
- `__IO uint32_t ProdSN`
- `__IO uint8_t Reserved1`
- `__IO uint16_t ManufactDate`
- `__IO uint8_t CID_CRC`
- `__IO uint8_t Reserved2`

#### Field Documentation

- `__IO uint8_t HAL_SD_CardCIDTypeDef::ManufacturerID`  
Manufacturer ID
- `__IO uint16_t HAL_SD_CardCIDTypeDef::OEM_AppliID`  
OEM/Application ID
- `__IO uint32_t HAL_SD_CardCIDTypeDef::ProdName1`  
Product Name part1
- `__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2`  
Product Name part2
- `__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev`  
Product Revision
- `__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN`  
Product Serial Number
- `__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1`  
Reserved1
- `__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate`  
Manufacturing Date
- `__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC`  
CID CRC
- `__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2`  
Always 1

### 61.1.5 HAL\_SD\_CardStatusTypeDef

*HAL\_SD\_CardStatusTypeDef* is defined in the `stm32f4xx_hal_sd.h`

#### Data Fields

- `__IO uint8_t DataBusWidth`
- `__IO uint8_t SecuredMode`
- `__IO uint16_t CardType`
- `__IO uint32_t ProtectedAreaSize`
- `__IO uint8_t SpeedClass`
- `__IO uint8_t PerformanceMove`
- `__IO uint8_t AllocationUnitSize`
- `__IO uint16_t EraseSize`
- `__IO uint8_t EraseTimeout`
- `__IO uint8_t EraseOffset`

#### Field Documentation

- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::DataBusWidth**  
Shows the currently defined data bus width
- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::SecuredMode**  
Card is in secured mode of operation
- **\_\_IO uint16\_t HAL\_SD\_CardStatusTypeDef::CardType**  
Carries information about card type
- **\_\_IO uint32\_t HAL\_SD\_CardStatusTypeDef::ProtectedAreaSize**  
Carries information about the capacity of protected area
- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::SpeedClass**  
Carries information about the speed class of the card
- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::PerformanceMove**  
Carries information about the card's performance move
- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::AllocationUnitSize**  
Carries information about the card's allocation unit size
- **\_\_IO uint16\_t HAL\_SD\_CardStatusTypeDef::EraseSize**  
Determines the number of AUs to be erased in one operation
- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::EraseTimeout**  
Determines the timeout for any number of AU erase
- **\_\_IO uint8\_t HAL\_SD\_CardStatusTypeDef::EraseOffset**  
Carries information about the erase offset

## 61.2 SD Firmware driver API description

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

### 61.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDIO and GPIO) are performed by the user in HAL\_SD\_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDIO memories which uses the HAL SDIO driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDIO low level resources by implementing the HAL\_SD\_MspInit() API:
  - a. Enable the SDIO interface clock using `__HAL_RCC_SDIO_CLK_ENABLE()`;
  - b. SDIO pins configuration for SD card
    - Enable the clock for the SDIO GPIOs using the functions `__HAL_RCC_GPIOx_CLK_ENABLE()`;
    - Configure these SDIO pins as alternate function pull-up using `HAL_GPIO_Init()` and according to your pin assignment;
  - c. DMA configuration if you need to use DMA process (`HAL_SD_ReadBlocks_DMA()` and `HAL_SD_WriteBlocks_DMA()` APIs).
    - Enable the DMAx interface clock using `__HAL_RCC_DMAx_CLK_ENABLE()`;
    - Configure the DMA using the function `HAL_DMA_Init()` with predeclared and filled.
  - d. NVIC configuration if you need to use interrupt process when using DMA transfer.
    - Configure the SDIO and DMA interrupt priorities using functions `HAL_NVIC_SetPriority()`; DMA priority is superior to SDIO's priority
    - Enable the NVIC DMA and SDIO IRQs using function `HAL_NVIC_EnableIRQ()`
    - SDIO interrupts are managed using the macros `__HAL_SD_ENABLE_IT()` and `__HAL_SD_DISABLE_IT()` inside the communication process.
    - SDIO interrupts pending bits are managed using the macros `__HAL_SD_GET_IT()` and `__HAL_SD_CLEAR_IT()`
  - e. NVIC configuration if you need to use interrupt process (`HAL_SD_ReadBlocks_IT()` and `HAL_SD_WriteBlocks_IT()` APIs).
    - Configure the SDIO interrupt priorities using function `HAL_NVIC_SetPriority()`;
    - Enable the NVIC SDIO IRQs using function `HAL_NVIC_EnableIRQ()`
    - SDIO interrupts are managed using the macros `__HAL_SD_ENABLE_IT()` and `__HAL_SD_DISABLE_IT()` inside the communication process.
    - SDIO interrupts pending bits are managed using the macros `__HAL_SD_GET_IT()` and `__HAL_SD_CLEAR_IT()`
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

### SD Card Initialization and configuration

To initialize the SD Card, use the `HAL_SD_Init()` function. It Initializes SDIO Peripheral(STM32 side) and the SD Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (`SDIO_CK`) is computed as follows:  $SDIO\_CK = SDIOCLK / (ClockDiv + 2)$  In initialization mode and according to the SD Card standard, make sure that the `SDIO_CK` frequency doesn't exceed 400KHz. This phase of initialization is done through `SDIO_Init()` and `SDIO_PowerState_ON()` SDIO low level APIs.
2. Initialize the SD card. The API used is `HAL_SD_InitCard()`. This phase allows the card initialization and identification and check the SD Card type (Standard Capacity or High Capacity) The initialization flow is compatible with SD standard. This API (`HAL_SD_InitCard()`) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the SD Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the `SDIO_CK` frequency doesn't exceed 25MHz and 50MHz in High-speed mode switch. To be able to use a frequency higher than 24MHz, you should use the SDIO peripheral in bypass mode. Refer to the corresponding reference manual for more details.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

### SD Card Read operation

- You can read from SD card in polling mode by using function `HAL_SD_ReadBlocks()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through `HAL_SD_GetCardState()` function for SD card state.

- You can read from SD card in DMA mode by using function HAL\_SD\_ReadBlocks\_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL\_SD\_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Rx interrupt event.
- You can read from SD card in Interrupt mode by using function HAL\_SD\_ReadBlocks\_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL\_SD\_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Rx interrupt event.

### SD Card Write operation

- You can write to SD card in polling mode by using function HAL\_SD\_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL\_SD\_GetCardState() function for SD card state.
- You can write to SD card in DMA mode by using function HAL\_SD\_WriteBlocks\_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL\_SD\_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Tx interrupt event.
- You can write to SD card in Interrupt mode by using function HAL\_SD\_WriteBlocks\_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL\_SD\_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Tx interrupt event.

### SD card status

- The SD Status contains status bits that are related to the SD Memory Card proprietary features. To get SD card status use the HAL\_SD\_GetCardStatus().

### SD card information

- To get SD card information, you can use the function HAL\_SD\_GetCardInfo(). It returns useful information about the SD card such as block size, card type, block number ...

### SD card CSD register

### SD card CID register

### SD HAL driver macros list

*Note:* You can refer to the SD HAL driver header file for more useful macros

### Callback registration

The compilation define USE\_HAL\_SD\_REGISTER\_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL\_SD\_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- MsplnitCallback : SD Msplnit.

- **MspDeInitCallback** : SD 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\_SD\_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- **TxCpltCallback** : callback when a transmission transfer is completed.
- **RxCpltCallback** : callback when a reception transfer is completed.
- **ErrorCallback** : callback when error occurs.
- **AbortCpltCallback** : callback when abort is completed.
- **MspInitCallback** : SD MspInit.
- **MspDeInitCallback** : SD MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL\_SD\_Init and if the state is HAL\_SD\_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\_SD\_Init and @ref HAL\_SD\_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL\_SD\_Init and @ref HAL\_SD\_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\_SD\_RegisterCallback before calling @ref HAL\_SD\_DeInit or @ref HAL\_SD\_Init function. When The compilation define USE\_HAL\_SD\_REGISTER\_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

## 61.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- [\*HAL\\_SD\\_Init\(\)\*](#)
- [\*HAL\\_SD\\_InitCard\(\)\*](#)
- [\*HAL\\_SD\\_DeInit\(\)\*](#)
- [\*HAL\\_SD\\_MspInit\(\)\*](#)
- [\*HAL\\_SD\\_MspDeInit\(\)\*](#)

## 61.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- [\*HAL\\_SD\\_ReadBlocks\(\)\*](#)
- [\*HAL\\_SD\\_WriteBlocks\(\)\*](#)
- [\*HAL\\_SD\\_ReadBlocks\\_IT\(\)\*](#)
- [\*HAL\\_SD\\_WriteBlocks\\_IT\(\)\*](#)
- [\*HAL\\_SD\\_ReadBlocks\\_DMA\(\)\*](#)
- [\*HAL\\_SD\\_WriteBlocks\\_DMA\(\)\*](#)
- [\*HAL\\_SD\\_Erase\(\)\*](#)
- [\*HAL\\_SD\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SD\\_GetState\(\)\*](#)
- [\*HAL\\_SD\\_GetError\(\)\*](#)
- [\*HAL\\_SD\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_SD\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_SD\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_SD\\_AbortCallback\(\)\*](#)

## 61.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations and get the related information

This section contains the following APIs:

- *HAL\_SD\_GetCardCID()*
- *HAL\_SD\_GetCardCSD()*
- *HAL\_SD\_GetCardStatus()*
- *HAL\_SD\_GetCardInfo()*
- *HAL\_SD\_ConfigWideBusOperation()*
- *HAL\_SD\_GetCardState()*
- *HAL\_SD\_Abort()*
- *HAL\_SD\_Abort\_IT()*

## 61.2.5 Detailed description of functions

### HAL\_SD\_Init

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_Init (SD\_HandleTypeDef \* hsd)**

#### Function description

Initializes the SD according to the specified parameters in the SD\_HandleTypeDef and create the associated handle.

#### Parameters

- **hsd**: Pointer to the SD handle

#### Return values

- **HAL**: status

### HAL\_SD\_InitCard

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_InitCard (SD\_HandleTypeDef \* hsd)**

#### Function description

Initializes the SD Card.

#### Parameters

- **hsd**: Pointer to SD handle

#### Return values

- **HAL**: status

#### Notes

- This function initializes the SD card. It could be used when a card re-initialization is needed.

### HAL\_SD\_DeInit

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_DeInit (SD\_HandleTypeDef \* hsd)**

#### Function description

De-Initializes the SD card.

#### Parameters

- **hsd**: Pointer to SD handle

#### Return values

- **HAL**: status

### HAL\_SD\_Msplnit

#### Function name

**void HAL\_SD\_Msplnit (SD\_HandleTypeDef \* hsd)**

#### Function description

Initializes the SD MSP.

#### Parameters

- **hsd**: Pointer to SD handle

#### Return values

- **None**:

### HAL\_SD\_MspDeInit

#### Function name

**void HAL\_SD\_MspDeInit (SD\_HandleTypeDef \* hsd)**

#### Function description

De-Initialize SD MSP.

#### Parameters

- **hsd**: Pointer to SD handle

#### Return values

- **None**:

### HAL\_SD\_ReadBlocks

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_ReadBlocks (SD\_HandleTypeDef \* hsd, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks, uint32\_t Timeout)**

#### Function description

Reads block(s) from a specified address in a card.

#### Parameters

- **hsd**: Pointer to SD handle
- **pData**: pointer to the buffer that will contain the received data
- **BlockAdd**: Block Address from where data is to be read
- **NumberOfBlocks**: Number of SD blocks to read
- **Timeout**: Specify timeout value

#### Return values

- **HAL**: status

#### Notes

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().

### HAL\_SD\_WriteBlocks

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_WriteBlocks (SD\_HandleTypeDef \* hsd, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks, uint32\_t Timeout)**



### Function description

Allows to write block(s) to a specified address in a card.

### Parameters

- **hsd**: Pointer to SD handle
- **pData**: pointer to the buffer that will contain the data to transmit
- **BlockAdd**: Block Address where data will be written
- **NumberOfBlocks**: Number of SD blocks to write
- **Timeout**: Specify timeout value

### Return values

- **HAL**: status

### Notes

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().

## HAL\_SD\_Erase

### Function name

**HAL\_StatusTypeDef HAL\_SD\_Erase (SD\_HandleTypeDef \* hsd, uint32\_t BlockStartAdd, uint32\_t BlockEndAdd)**

### Function description

Erases the specified memory area of the given SD card.

### Parameters

- **hsd**: Pointer to SD handle
- **BlockStartAdd**: Start Block address
- **BlockEndAdd**: End Block address

### Return values

- **HAL**: status

### Notes

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().

## HAL\_SD\_ReadBlocks\_IT

### Function name

**HAL\_StatusTypeDef HAL\_SD\_ReadBlocks\_IT (SD\_HandleTypeDef \* hsd, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

### Function description

Reads block(s) from a specified address in a card.

### Parameters

- **hsd**: Pointer to SD handle
- **pData**: Pointer to the buffer that will contain the received data
- **BlockAdd**: Block Address from where data is to be read
- **NumberOfBlocks**: Number of blocks to read.

### Return values

- **HAL**: status

### Notes

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().
- You could also check the IT transfer process through the SD Rx interrupt event.

### HAL\_SD\_WriteBlocks\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_WriteBlocks\_IT (SD\_HandleTypeDef \* hsd, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Writes block(s) to a specified address in a card.

#### Parameters

- **hsd**: Pointer to SD handle
- **pData**: Pointer to the buffer that will contain the data to transmit
- **BlockAdd**: Block Address where data will be written
- **NumberOfBlocks**: Number of blocks to write

#### Return values

- **HAL**: status

#### Notes

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().
- You could also check the IT transfer process through the SD Tx interrupt event.

### HAL\_SD\_ReadBlocks\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_ReadBlocks\_DMA (SD\_HandleTypeDef \* hsd, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Reads block(s) from a specified address in a card.

#### Parameters

- **hsd**: Pointer SD handle
- **pData**: Pointer to the buffer that will contain the received data
- **BlockAdd**: Block Address from where data is to be read
- **NumberOfBlocks**: Number of blocks to read.

#### Return values

- **HAL**: status

#### Notes

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().
- You could also check the DMA transfer process through the SD Rx interrupt event.

### HAL\_SD\_WriteBlocks\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_WriteBlocks\_DMA (SD\_HandleTypeDef \* hsd, uint8\_t \* pData, uint32\_t BlockAdd, uint32\_t NumberOfBlocks)**

#### Function description

Writes block(s) to a specified address in a card.

**Parameters**

- **hsd**: Pointer to SD handle
- **pData**: Pointer to the buffer that will contain the data to transmit
- **BlockAdd**: Block Address where data will be written
- **NumberOfBlocks**: Number of blocks to write

**Return values**

- **HAL**: status

**Notes**

- This API should be followed by a check on the card state through HAL\_SD\_GetCardState().
- You could also check the DMA transfer process through the SD Tx interrupt event.

**HAL\_SD\_IRQHandler**
**Function name**

```
void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)
```

**Function description**

This function handles SD card interrupt request.

**Parameters**

- **hsd**: Pointer to SD handle

**Return values**

- **None**:

**HAL\_SD\_TxCpltCallback**
**Function name**

```
void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)
```

**Function description**

Tx Transfer completed callbacks.

**Parameters**

- **hsd**: Pointer to SD handle

**Return values**

- **None**:

**HAL\_SD\_RxCpltCallback**
**Function name**

```
void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)
```

**Function description**

Rx Transfer completed callbacks.

**Parameters**

- **hsd**: Pointer SD handle

**Return values**

- **None**:

### HAL\_SD\_ErrorCallback

#### Function name

**void HAL\_SD\_ErrorCallback (SD\_HandleTypeDef \* hsd)**

#### Function description

SD error callbacks.

#### Parameters

- **hsd**: Pointer SD handle

#### Return values

- **None**:

### HAL\_SD\_AbortCallback

#### Function name

**void HAL\_SD\_AbortCallback (SD\_HandleTypeDef \* hsd)**

#### Function description

SD Abort callbacks.

#### Parameters

- **hsd**: Pointer SD handle

#### Return values

- **None**:

### HAL\_SD\_ConfigWideBusOperation

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_ConfigWideBusOperation (SD\_HandleTypeDef \* hsd, uint32\_t WideMode)**

#### Function description

Enables wide bus operation for the requested card if supported by card.

#### Parameters

- **hsd**: Pointer to SD handle
- **WideMode**: Specifies the SD card wide bus mode This parameter can be one of the following values:
  - SDIO\_BUS\_WIDE\_8B: 8-bit data transfer
  - SDIO\_BUS\_WIDE\_4B: 4-bit data transfer
  - SDIO\_BUS\_WIDE\_1B: 1-bit data transfer

#### Return values

- **HAL**: status

### HAL\_SD\_SendSDStatus

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_SendSDStatus (SD\_HandleTypeDef \* hsd, uint32\_t \* pSDstatus)**

#### Function description

### HAL\_SD\_GetCardState

#### Function name

**HAL\_SD\_CardStateTypeDef HAL\_SD\_GetCardState (SD\_HandleTypeDef \* hsd)**

**Function description**

Gets the current sd card data state.

**Parameters**

- **hsd**: pointer to SD handle

**Return values**

- **Card**: state

**HAL\_SD\_GetCardCID**
**Function name**

**HAL\_StatusTypeDef HAL\_SD\_GetCardCID (SD\_HandleTypeDef \* hsd, HAL\_SD\_CardCIDTypeDef \* pCID)**

**Function description**

Returns information the information of the card which are stored on the CID register.

**Parameters**

- **hsd**: Pointer to SD handle
- **pCID**: Pointer to a HAL\_SD\_CardCIDTypeDef structure that contains all CID register parameters

**Return values**

- **HAL**: status

**HAL\_SD\_GetCardCSD**
**Function name**

**HAL\_StatusTypeDef HAL\_SD\_GetCardCSD (SD\_HandleTypeDef \* hsd, HAL\_SD\_CardCSDTypeDef \* pCSD)**

**Function description**

Returns information the information of the card which are stored on the CSD register.

**Parameters**

- **hsd**: Pointer to SD handle
- **pCSD**: Pointer to a HAL\_SD\_CardCSDTypeDef structure that contains all CSD register parameters

**Return values**

- **HAL**: status

**HAL\_SD\_GetCardStatus**
**Function name**

**HAL\_StatusTypeDef HAL\_SD\_GetCardStatus (SD\_HandleTypeDef \* hsd, HAL\_SD\_CardStatusTypeDef \* pStatus)**

**Function description**

Gets the SD status info.

**Parameters**

- **hsd**: Pointer to SD handle
- **pStatus**: Pointer to the HAL\_SD\_CardStatusTypeDef structure that will contain the SD card status information

**Return values**

- **HAL**: status

### HAL\_SD\_GetCardInfo

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_GetCardInfo (SD\_HandleTypeDef \* hsd, HAL\_SD\_CardInfoTypeDef \* pCardInfo)**

#### Function description

Gets the SD card info.

#### Parameters

- **hsd**: Pointer to SD handle
- **pCardInfo**: Pointer to the HAL\_SD\_CardInfoTypeDef structure that will contain the SD card status information

#### Return values

- **HAL**: status

### HAL\_SD\_GetState

#### Function name

**HAL\_SD\_StateTypeDef HAL\_SD\_GetState (SD\_HandleTypeDef \* hsd)**

#### Function description

return the SD state

#### Parameters

- **hsd**: Pointer to sd handle

#### Return values

- **HAL**: state

### HAL\_SD\_GetError

#### Function name

**uint32\_t HAL\_SD\_GetError (SD\_HandleTypeDef \* hsd)**

#### Function description

Return the SD error code.

#### Parameters

- **hsd**: : Pointer to a SD\_HandleTypeDef structure that contains the configuration information.

#### Return values

- **SD**: Error Code

### HAL\_SD\_Abort

#### Function name

**HAL\_StatusTypeDef HAL\_SD\_Abort (SD\_HandleTypeDef \* hsd)**

#### Function description

Abort the current transfer and disable the SD.

#### Parameters

- **hsd**: pointer to a SD\_HandleTypeDef structure that contains the configuration information for SD module.

#### Return values

- **HAL**: status

## HAL\_SD\_Abort\_IT

### Function name

HAL\_StatusTypeDef HAL\_SD\_Abort\_IT (SD\_HandleTypeDef \* hsd)

### Function description

Abort the current transfer and disable the SD (IT mode).

### Parameters

- **hsd**: pointer to a SD\_HandleTypeDef structure that contains the configuration information for SD module.

### Return values

- **HAL**: status

## 61.3 SD Firmware driver defines

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

### 61.3.1 SD

SD

*SD Error status enumeration Structure definition*

#### HAL\_SD\_ERROR\_NONE

No error

#### HAL\_SD\_ERROR\_CMD\_CRC\_FAIL

Command response received (but CRC check failed)

#### HAL\_SD\_ERROR\_DATA\_CRC\_FAIL

Data block sent/received (CRC check failed)

#### HAL\_SD\_ERROR\_CMD\_RSP\_TIMEOUT

Command response timeout

#### HAL\_SD\_ERROR\_DATA\_TIMEOUT

Data timeout

#### HAL\_SD\_ERROR\_TX\_UNDERRUN

Transmit FIFO underrun

#### HAL\_SD\_ERROR\_RX\_OVERRUN

Receive FIFO overrun

#### HAL\_SD\_ERROR\_ADDR\_MISALIGNED

Misaligned address

#### HAL\_SD\_ERROR\_BLOCK\_LEN\_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

#### HAL\_SD\_ERROR\_ERASE\_SEQ\_ERR

An error in the sequence of erase command occurs

#### HAL\_SD\_ERROR\_BAD\_ERASE\_PARAM

An invalid selection for erase groups

**HAL\_SD\_ERROR\_WRITE\_PROT\_VIOLATION**

Attempt to program a write protect block

**HAL\_SD\_ERROR\_LOCK\_UNLOCK\_FAILED**

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

**HAL\_SD\_ERROR\_COM\_CRC\_FAILED**

CRC check of the previous command failed

**HAL\_SD\_ERROR\_ILLEGAL\_CMD**

Command is not legal for the card state

**HAL\_SD\_ERROR\_CARD\_ECC\_FAILED**

Card internal ECC was applied but failed to correct the data

**HAL\_SD\_ERROR\_CC\_ERR**

Internal card controller error

**HAL\_SD\_ERROR\_GENERAL\_UNKNOWN\_ERR**

General or unknown error

**HAL\_SD\_ERROR\_STREAM\_READ\_UNDERRUN**

The card could not sustain data reading in stream mode

**HAL\_SD\_ERROR\_STREAM\_WRITE\_OVERRUN**

The card could not sustain data programming in stream mode

**HAL\_SD\_ERROR\_CID\_CSD\_OVERWRITE**

CID/CSD overwrite error

**HAL\_SD\_ERROR\_WP\_ERASE\_SKIP**

Only partial address space was erased

**HAL\_SD\_ERROR\_CARD\_ECC\_DISABLED**

Command has been executed without using internal ECC

**HAL\_SD\_ERROR\_ERASE\_RESET**

Erase sequence was cleared before executing because an out of erase sequence command was received

**HAL\_SD\_ERROR\_AKE\_SEQ\_ERR**

Error in sequence of authentication

**HAL\_SD\_ERROR\_INVALID\_VOLTRANGE**

Error in case of invalid voltage range

**HAL\_SD\_ERROR\_ADDR\_OUT\_OF\_RANGE**

Error when addressed block is out of range

**HAL\_SD\_ERROR\_REQUEST\_NOT\_APPLICABLE**

Error when command request is not applicable

**HAL\_SD\_ERROR\_PARAM**

the used parameter is not valid

**HAL\_SD\_ERROR\_UNSUPPORTED\_FEATURE**

Error when feature is not insupported



**HAL\_SD\_ERROR\_BUSY**

Error when transfer process is busy

**HAL\_SD\_ERROR\_DMA**

Error while DMA transfer

**HAL\_SD\_ERROR\_TIMEOUT**

Timeout error

***SD context enumeration*****SD\_CONTEXT\_NONE**

None

**SD\_CONTEXT\_READ\_SINGLE\_BLOCK**

Read single block operation

**SD\_CONTEXT\_READ\_MULTIPLE\_BLOCK**

Read multiple blocks operation

**SD\_CONTEXT\_WRITE\_SINGLE\_BLOCK**

Write single block operation

**SD\_CONTEXT\_WRITE\_MULTIPLE\_BLOCK**

Write multiple blocks operation

**SD\_CONTEXT\_IT**

Process in Interrupt mode

**SD\_CONTEXT\_DMA**

Process in DMA mode

***SD Supported Memory Cards*****CARD\_SDSC**

SD Standard Capacity <2Go

**CARD\_SDHC\_SDXC**

SD High Capacity <32Go, SD Extended Capacity <2To

**CARD\_SECURED*****SD Supported Version*****CARD\_V1\_X****CARD\_V2\_X*****Exported Constants*****BLOCKSIZE**

Block size is 512 bytes

***SD Exported Macros***

### **\_\_HAL\_SD\_RESET\_HANDLE\_STATE**

**Description:**

- Reset SD handle state.

**Parameters:**

- `__HANDLE__`: SD handle.

**Return value:**

- None

### **\_\_HAL\_SD\_ENABLE**

**Description:**

- Enable the SD device.

**Return value:**

- None

### **\_\_HAL\_SD\_DISABLE**

**Description:**

- Disable the SD device.

**Return value:**

- None

### **\_\_HAL\_SD\_DMA\_ENABLE**

**Description:**

- Enable the SDMMC DMA transfer.

**Return value:**

- None

### **\_\_HAL\_SD\_DMA\_DISABLE**

**Description:**

- Disable the SDMMC DMA transfer.

**Return value:**

- None

## \_\_HAL\_SD\_ENABLE\_IT

### Description:

- Enable the SD device interrupt.

### Parameters:

- `__HANDLE__`: SD Handle
- `__INTERRUPT__`: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRCFail`: Data block sent/received (CRC check failed) interrupt
  - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
  - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
  - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
  - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
  - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
  - `SDIO_IT_CMDSSENT`: Command sent (no response required) interrupt
  - `SDIO_IT_DATAEND`: Data end (data counter, `DATA_COUNT`, is zero) interrupt
  - `SDIO_IT_DBCKEND`: Data block sent/received (CRC check passed) interrupt
  - `SDIO_IT_CMDACT`: Command transfer in progress interrupt
  - `SDIO_IT_TXACT`: Data transmit in progress interrupt
  - `SDIO_IT_RXACT`: Data receive in progress interrupt
  - `SDIO_IT_TXFIFOHE`: Transmit FIFO Half Empty interrupt
  - `SDIO_IT_RXFIFOHF`: Receive FIFO Half Full interrupt
  - `SDIO_IT_TXFIFO`: Transmit FIFO full interrupt
  - `SDIO_IT_RXFIFO`: Receive FIFO full interrupt
  - `SDIO_IT_TXFIFOE`: Transmit FIFO empty interrupt
  - `SDIO_IT_RXFIFOE`: Receive FIFO empty interrupt
  - `SDIO_IT_TXDAVL`: Data available in transmit FIFO interrupt
  - `SDIO_IT_RXDAVL`: Data available in receive FIFO interrupt
  - `SDIO_IT_SDIOIT`: SDIO interrupt received interrupt

### Return value:

- None

## \_\_HAL\_SD\_DISABLE\_IT

### Description:

- Disable the SD device interrupt.

### Parameters:

- `__HANDLE__`: SD Handle
- `__INTERRUPT__`: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRRCFAIL`: Data block sent/received (CRC check failed) interrupt
  - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
  - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
  - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
  - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
  - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
  - `SDIO_IT_CMDSSENT`: Command sent (no response required) interrupt
  - `SDIO_IT_DATAEND`: Data end (data counter, `DATACOUNT`, is zero) interrupt
  - `SDIO_IT_DBCKEND`: Data block sent/received (CRC check passed) interrupt
  - `SDIO_IT_CMDACT`: Command transfer in progress interrupt
  - `SDIO_IT_TXACT`: Data transmit in progress interrupt
  - `SDIO_IT_RXACT`: Data receive in progress interrupt
  - `SDIO_IT_TXFIFOHE`: Transmit FIFO Half Empty interrupt
  - `SDIO_IT_RXFIFOHF`: Receive FIFO Half Full interrupt
  - `SDIO_IT_TXFIFO`: Transmit FIFO full interrupt
  - `SDIO_IT_RXFIFO`: Receive FIFO full interrupt
  - `SDIO_IT_TXFIFOE`: Transmit FIFO empty interrupt
  - `SDIO_IT_RXFIFOE`: Receive FIFO empty interrupt
  - `SDIO_IT_TXDAVL`: Data available in transmit FIFO interrupt
  - `SDIO_IT_RXDAVL`: Data available in receive FIFO interrupt
  - `SDIO_IT_SDIOIT`: SDIO interrupt received interrupt

### Return value:

- None

## \_\_HAL\_SD\_GET\_FLAG

### Description:

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

### Parameters:

- \_\_HANDLE\_\_: SD Handle
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - SDIO\_FLAG\_CCRCFAIL: Command response received (CRC check failed)
  - SDIO\_FLAG\_DCRRCFAIL: Data block sent/received (CRC check failed)
  - SDIO\_FLAG\_CTIMEOUT: Command response timeout
  - SDIO\_FLAG\_DTIMEOUT: Data timeout
  - SDIO\_FLAG\_TXUNDERR: Transmit FIFO underrun error
  - SDIO\_FLAG\_RXOVERR: Received FIFO overrun error
  - SDIO\_FLAG\_CMDREND: Command response received (CRC check passed)
  - SDIO\_FLAG\_CMDSSENT: Command sent (no response required)
  - SDIO\_FLAG\_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDIO\_FLAG\_DBCKEND: Data block sent/received (CRC check passed)
  - SDIO\_FLAG\_CMDACT: Command transfer in progress
  - SDIO\_FLAG\_TXACT: Data transmit in progress
  - SDIO\_FLAG\_RXACT: Data receive in progress
  - SDIO\_FLAG\_TXFIFOHE: Transmit FIFO Half Empty
  - SDIO\_FLAG\_RXFIFOHF: Receive FIFO Half Full
  - SDIO\_FLAG\_TXFIFO: Transmit FIFO full
  - SDIO\_FLAG\_RXFIFO: Receive FIFO full
  - SDIO\_FLAG\_TXFIFOE: Transmit FIFO empty
  - SDIO\_FLAG\_RXFIFOE: Receive FIFO empty
  - SDIO\_FLAG\_TXDAVL: Data available in transmit FIFO
  - SDIO\_FLAG\_RXDAVL: Data available in receive FIFO
  - SDIO\_FLAG\_SDIOIT: SDIO interrupt received

### Return value:

- The: new state of SD FLAG (SET or RESET).

## \_\_HAL\_SD\_CLEAR\_FLAG

**Description:**

- Clear the SD's pending flags.

**Parameters:**

- `__HANDLE__`: SD Handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
  - `SDIO_FLAG_DCRFAIL`: Data block sent/received (CRC check failed)
  - `SDIO_FLAG_CTIMEOUT`: Command response timeout
  - `SDIO_FLAG_DTIMEOUT`: Data timeout
  - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
  - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
  - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
  - `SDIO_FLAG_CMDSSENT`: Command sent (no response required)
  - `SDIO_FLAG_DATAEND`: Data end (data counter, `DATACOUNT`, is zero)
  - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
  - `SDIO_FLAG_SDIOIT`: SDIO interrupt received

**Return value:**

- None

## \_\_HAL\_SD\_GET\_IT

### Description:

- Check whether the specified SD interrupt has occurred or not.

### Parameters:

- `__HANDLE__`: SD Handle
- `__INTERRUPT__`: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRFAIL`: Data block sent/received (CRC check failed) interrupt
  - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
  - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
  - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
  - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
  - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
  - `SDIO_IT_CMDSSENT`: Command sent (no response required) interrupt
  - `SDIO_IT_DATAEND`: Data end (data counter, `DATA_COUNT`, is zero) interrupt
  - `SDIO_IT_DBCKEND`: Data block sent/received (CRC check passed) interrupt
  - `SDIO_IT_CMDACT`: Command transfer in progress interrupt
  - `SDIO_IT_TXACT`: Data transmit in progress interrupt
  - `SDIO_IT_RXACT`: Data receive in progress interrupt
  - `SDIO_IT_TXFIFOHE`: Transmit FIFO Half Empty interrupt
  - `SDIO_IT_RXFIFOHF`: Receive FIFO Half Full interrupt
  - `SDIO_IT_TXFIFO`: Transmit FIFO full interrupt
  - `SDIO_IT_RXFIFO`: Receive FIFO full interrupt
  - `SDIO_IT_TXFIFOE`: Transmit FIFO empty interrupt
  - `SDIO_IT_RXFIFOE`: Receive FIFO empty interrupt
  - `SDIO_IT_TXDAVL`: Data available in transmit FIFO interrupt
  - `SDIO_IT_RXDAVL`: Data available in receive FIFO interrupt
  - `SDIO_IT_SDIOIT`: SDIO interrupt received interrupt

### Return value:

- The: new state of SD IT (SET or RESET).

## \_\_HAL\_SD\_CLEAR\_IT

### Description:

- Clear the SD's interrupt pending bits.

### Parameters:

- \_\_HANDLE\_\_: SD Handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDIO\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO\_IT\_DCRRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDIO\_IT\_DTIMEOUT: Data timeout interrupt
  - SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt
  - SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
  - SDIO\_IT\_CMDSSENT: Command sent (no response required) interrupt
  - SDIO\_IT\_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDIO\_IT\_SDIOIT: SDIO interrupt received interrupt

### Return value:

- None

### *SD Card State enumeration structure*

#### HAL\_SD\_CARD\_READY

Card state is ready

#### HAL\_SD\_CARD\_IDENTIFICATION

Card is in identification state

#### HAL\_SD\_CARD\_STANDBY

Card is in standby state

#### HAL\_SD\_CARD\_TRANSFER

Card is in transfer state

#### HAL\_SD\_CARD\_SENDING

Card is sending an operation

#### HAL\_SD\_CARD\_RECEIVING

Card is receiving operation information

#### HAL\_SD\_CARD\_PROGRAMMING

Card is in programming state

#### HAL\_SD\_CARD\_DISCONNECTED

Card is disconnected

#### HAL\_SD\_CARD\_ERROR

Card response Error

### *SD Handle Structure definition*

#### SD\_InitTypeDef

#### SD\_TypeDef



## 62 HAL SDRAM Generic Driver

### 62.1 SDRAM Firmware driver registers structures

#### 62.1.1 SDRAM\_HandleTypeDef

*SDRAM\_HandleTypeDef* is defined in the stm32f4xx\_hal\_sdram.h

##### Data Fields

- *FMC\_SDRAM\_TypeDef \* Instance*
- *FMC\_SDRAM\_InitTypeDef Init*
- *\_\_IO HAL\_SDRAM\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*
- *DMA\_HandleTypeDef \* hdma*

##### Field Documentation

- *FMC\_SDRAM\_TypeDef\* SDRAM\_HandleTypeDef::Instance*  
Register base address
- *FMC\_SDRAM\_InitTypeDef SDRAM\_HandleTypeDef::Init*  
SDRAM device configuration parameters
- *\_\_IO HAL\_SDRAM\_StateTypeDef SDRAM\_HandleTypeDef::State*  
SDRAM access state
- *HAL\_LockTypeDef SDRAM\_HandleTypeDef::Lock*  
SDRAM locking object
- *DMA\_HandleTypeDef\* SDRAM\_HandleTypeDef::hdma*  
Pointer DMA handler

### 62.2 SDRAM Firmware driver API description

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

#### 62.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a *SDRAM\_HandleTypeDef* handle structure, for example: *SDRAM\_HandleTypeDef* hdsram
  - Fill the *SDRAM\_HandleTypeDef* handle "Init" field with the allowed values of the structure member.
  - Fill the *SDRAM\_HandleTypeDef* handle "Instance" field with a predefined base register instance for NOR or SDRAM device
2. Declare a *FMC\_SDRAM\_TimingTypeDef* structure; for example: *FMC\_SDRAM\_TimingTypeDef* Timing; and fill its fields with the allowed values of the structure member.
3. Initialize the SDRAM Controller by calling the function *HAL\_SDRAM\_Init()*. This function performs the following sequence:
  - a. MSP hardware layer configuration using the function *HAL\_SDRAM\_MspInit()*
  - b. Control register configuration using the FMC SDRAM interface function *FMC\_SDRAM\_Init()*
  - c. Timing register configuration using the FMC SDRAM interface function *FMC\_SDRAM\_Timing\_Init()*
  - d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.
4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
  - *HAL\_SDRAM\_Read()/HAL\_SDRAM\_Write()* for polling read/write access
  - *HAL\_SDRAM\_Read\_DMA()/HAL\_SDRAM\_Write\_DMA()* for DMA read/write transfer

5. You can also control the SDRAM device by calling the control APIs `HAL_SDRAM_WriteOperation_Enable()`/`HAL_SDRAM_WriteOperation_Disable()` to respectively enable/disable the SDRAM write operation or the function `HAL_SDRAM_SendCommand()` to send a specified command to the SDRAM device. The command to be sent must be configured with the `FMC_SDRAM_CommandTypeDef` structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function `HAL_SDRAM_GetState()`

### Callback registration

The compilation define `USE_HAL_SDRAM_REGISTER_CALLBACKS` when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions `@ref HAL_SDRAM_RegisterCallback()` to register a user callback, it allows to register following callbacks:

- `MspInitCallback` : SDRAM `MspInit`.
- `MspDeInitCallback` : SDRAM `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_SDRAM_UnRegisterCallback()` to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- `MspInitCallback` : SDRAM `MspInit`.
- `MspDeInitCallback` : SDRAM `MspDeInit`. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the `@ref HAL_SDRAM_Init` and if the state is `HAL_SDRAM_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_SDRAM_Init` and `@ref HAL_SDRAM_DeInit` only when these callbacks are null (not registered beforehand). If not, `MspInit` or `MspDeInit` are not null, the `@ref HAL_SDRAM_Init` and `@ref HAL_SDRAM_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_SDRAM_RegisterCallback` before calling `@ref HAL_SDRAM_DeInit` or `@ref HAL_SDRAM_Init` function. When The compilation define `USE_HAL_SDRAM_REGISTER_CALLBACKS` is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

## 62.2.2 SDRAM Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

This section contains the following APIs:

- [\*HAL\\_SDRAM\\_Init\(\)\*](#)
- [\*HAL\\_SDRAM\\_DeInit\(\)\*](#)
- [\*HAL\\_SDRAM\\_MspInit\(\)\*](#)
- [\*HAL\\_SDRAM\\_MspDeInit\(\)\*](#)
- [\*HAL\\_SDRAM\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SDRAM\\_RefreshErrorCallback\(\)\*](#)
- [\*HAL\\_SDRAM\\_DMA\\_XferCpltCallback\(\)\*](#)
- [\*HAL\\_SDRAM\\_DMA\\_XferErrorCallback\(\)\*](#)

## 62.2.3 SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

This section contains the following APIs:

- [\*HAL\\_SDRAM\\_Read\\_8b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_8b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Read\\_16b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_16b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Read\\_32b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_32b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Read\\_DMA\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_DMA\(\)\*](#)

### 62.2.4 SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

This section contains the following APIs:

- [HAL\\_SDRAM\\_WriteProtection\\_Enable\(\)](#)
- [HAL\\_SDRAM\\_WriteProtection\\_Disable\(\)](#)
- [HAL\\_SDRAM\\_SendCommand\(\)](#)
- [HAL\\_SDRAM\\_ProgramRefreshRate\(\)](#)
- [HAL\\_SDRAM\\_SetAutoRefreshNumber\(\)](#)
- [HAL\\_SDRAM\\_GetModeStatus\(\)](#)

### 62.2.5 SDRAM State functions

This subsection permits to get in run-time the status of the SDRAM controller and the data flow.

This section contains the following APIs:

- [HAL\\_SDRAM\\_GetState\(\)](#)

### 62.2.6 Detailed description of functions

#### HAL\_SDRAM\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Init (SDRAM\_HandleTypeDef \* hsdram, FMC\_SDRAM\_TimingTypeDef \* Timing)**

##### Function description

Performs the SDRAM device initialization sequence.

##### Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Timing**: Pointer to SDRAM control timing structure

##### Return values

- **HAL**: status

#### HAL\_SDRAM\_DeInit

##### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_DeInit (SDRAM\_HandleTypeDef \* hsdram)**

##### Function description

Perform the SDRAM device initialization sequence.

##### Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

##### Return values

- **HAL**: status

#### HAL\_SDRAM\_Msplnit

##### Function name

**void HAL\_SDRAM\_Msplnit (SDRAM\_HandleTypeDef \* hsdram)**

##### Function description

SDRAM MSP Init.

**Parameters**

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

**Return values**

- **None**:

**HAL\_SDRAM\_MspDeInit**
**Function name**

```
void HAL_SDRAM_MspDeInit (SDRAM_HandleTypeDef * hsdram)
```

**Function description**

SDRAM MSP DeInit.

**Parameters**

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

**Return values**

- **None**:

**HAL\_SDRAM\_IRQHandler**
**Function name**

```
void HAL_SDRAM_IRQHandler (SDRAM_HandleTypeDef * hsdram)
```

**Function description**

This function handles SDRAM refresh error interrupt request.

**Parameters**

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

**Return values**

- **HAL**: status

**HAL\_SDRAM\_RefreshErrorCallback**
**Function name**

```
void HAL_SDRAM_RefreshErrorCallback (SDRAM_HandleTypeDef * hsdram)
```

**Function description**

SDRAM Refresh error callback.

**Parameters**

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

**Return values**

- **None**:

**HAL\_SDRAM\_DMA\_XferCpltCallback**
**Function name**

```
void HAL_SDRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)
```

**Function description**

DMA transfer complete callback.

### Parameters

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA module.

### Return values

- **None:**

**HAL\_SDRAM\_DMA\_XferErrorCallback**

### Function name

**void HAL\_SDRAM\_DMA\_XferErrorCallback (DMA\_HandleTypeDef \* hdma)**

### Function description

DMA transfer complete error callback.

### Parameters

- **hdma:** DMA handle

### Return values

- **None:**

**HAL\_SDRAM\_Read\_8b**

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Read\_8b (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint8\_t \* pDstBuffer, uint32\_t BufferSize)**

### Function description

Reads 8-bit data buffer from the SDRAM memory.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Write\_8b**

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Write\_8b (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint8\_t \* pSrcBuffer, uint32\_t BufferSize)**

### Function description

Writes 8-bit data buffer to SDRAM memory.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Read\_16b

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Read\_16b** (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint16\_t \* pDstBuffer, uint32\_t BufferSize)

### Function description

Reads 16-bit data buffer from the SDRAM memory.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Write\_16b

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Write\_16b** (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint16\_t \* pSrcBuffer, uint32\_t BufferSize)

### Function description

Writes 16-bit data buffer to SDRAM memory.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Read\_32b

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Read\_32b** (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint32\_t \* pDstBuffer, uint32\_t BufferSize)

### Function description

Reads 32-bit data buffer from the SDRAM memory.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Write\_32b

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Write\_32b** (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint32\_t \* pSrcBuffer, uint32\_t BufferSize)

### Function description

Writes 32-bit data buffer to SDRAM memory.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Read\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Read\_DMA** (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint32\_t \* pDstBuffer, uint32\_t BufferSize)

### Function description

Reads a Words data from the SDRAM memory using DMA transfer.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_Write\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_Write\_DMA** (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint32\_t \* pSrcBuffer, uint32\_t BufferSize)

### Function description

Writes a Words data buffer to SDRAM memory using DMA transfer.

### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM 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\_SDRAM\_WriteProtection\_Enable**

#### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_WriteProtection\_Enable (SDRAM\_HandleTypeDef \* hsdram)**

#### Function description

Enables dynamically SDRAM write protection.

#### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

#### Return values

- **HAL:** status

**HAL\_SDRAM\_WriteProtection\_Disable**

#### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_WriteProtection\_Disable (SDRAM\_HandleTypeDef \* hsdram)**

#### Function description

Disables dynamically SDRAM write protection.

#### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

#### Return values

- **HAL:** status

**HAL\_SDRAM\_SendCommand**

#### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_SendCommand (SDRAM\_HandleTypeDef \* hsdram, FMC\_SDRAM\_CommandTypeDef \* Command, uint32\_t Timeout)**

#### Function description

Sends Command to the SDRAM bank.

#### Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Command:** SDRAM command structure
- **Timeout:** Timeout duration

#### Return values

- **HAL:** status

**HAL\_SDRAM\_ProgramRefreshRate**

#### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_ProgramRefreshRate (SDRAM\_HandleTypeDef \* hsdram, uint32\_t RefreshRate)**



### Function description

Programs the SDRAM Memory Refresh rate.

### Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **RefreshRate**: The SDRAM refresh rate value

### Return values

- **HAL**: status

### HAL\_SDRAM\_SetAutoRefreshNumber

### Function name

**HAL\_StatusTypeDef HAL\_SDRAM\_SetAutoRefreshNumber (SDRAM\_HandleTypeDef \* hsdram, uint32\_t AutoRefreshNumber)**

### Function description

Sets the Number of consecutive SDRAM Memory auto Refresh commands.

### Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **AutoRefreshNumber**: The SDRAM auto Refresh number

### Return values

- **HAL**: status

### HAL\_SDRAM\_GetModeStatus

### Function name

**uint32\_t HAL\_SDRAM\_GetModeStatus (SDRAM\_HandleTypeDef \* hsdram)**

### Function description

Returns the SDRAM memory current mode.

### Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

### Return values

- **The**: SDRAM memory mode.

### HAL\_SDRAM\_GetState

### Function name

**HAL\_SDRAM\_StateTypeDef HAL\_SDRAM\_GetState (SDRAM\_HandleTypeDef \* hsdram)**

### Function description

Returns the SDRAM state.

### Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

### Return values

- **HAL**: state

## 62.3 SDRAM Firmware driver defines

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

### 62.3.1 SDRAM

SDRAM

#### *SDRAM Exported Macros*

##### `__HAL_SDRAM_RESET_HANDLE_STATE`

**Description:**

- Reset SDRAM handle state.

**Parameters:**

- `__HANDLE__`: specifies the SDRAM handle.

**Return value:**

- None

## 63 HAL SMARTCARD Generic Driver

### 63.1 SMARTCARD Firmware driver registers structures

#### 63.1.1 SMARTCARD\_InitTypeDef

*SMARTCARD\_InitTypeDef* is defined in the `stm32f4xx_hal_smartcard.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 Prescaler*
- *uint32\_t GuardTime*
- *uint32\_t NACKState*

##### Field Documentation

- *uint32\_t SMARTCARD\_InitTypeDef::BaudRate*  
This member configures the SmartCard communication baud rate. The baud rate is computed using the following formula:
  - $IntegerDivisor = ((PCLKx) / (16 * (hsc->Init.BaudRate)))$
  - $FractionalDivisor = ((IntegerDivisor - ((uint32_t) IntegerDivisor)) * 16) + 0.5$
- *uint32\_t SMARTCARD\_InitTypeDef::WordLength*  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [SMARTCARD\\_Word\\_Length](#)
- *uint32\_t SMARTCARD\_InitTypeDef::StopBits*  
Specifies the number of stop bits transmitted. This parameter can be a value of [SMARTCARD\\_Stop\\_Bits](#)
- *uint32\_t SMARTCARD\_InitTypeDef::Parity*  
Specifies the parity mode. This parameter can be a value of [SMARTCARD\\_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 SMARTCARD\_InitTypeDef::Mode*  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [SMARTCARD\\_Mode](#)
- *uint32\_t SMARTCARD\_InitTypeDef::CLKPolarity*  
Specifies the steady state of the serial clock. This parameter can be a value of [SMARTCARD\\_Clock\\_Polarity](#)
- *uint32\_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](#)
- *uint32\_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](#)
- *uint32\_t SMARTCARD\_InitTypeDef::Prescaler*  
Specifies the SmartCard Prescaler value used for dividing the system clock to provide the smartcard clock. The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency. This parameter can be a value of [SMARTCARD\\_Prescaler](#)

- **`uint32_t SMARTCARD_InitTypeDef::GuardTime`**  
Specifies the SmartCard Guard Time value in terms of number of baud clocks
- **`uint32_t SMARTCARD_InitTypeDef::NACKState`**  
Specifies the SmartCard NACK Transmission state. This parameter can be a value of [SMARTCARD\\_NACK\\_State](#)

### 63.1.2 `__SMARTCARD_HandleTypeDef`

`__SMARTCARD_HandleTypeDef` is defined in the `stm32f4xx_hal_smartcard.h`

#### Data Fields

- **`USART_TypeDef * Instance`**
- **`SMARTCARD_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`__IO uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`__IO uint16_t RxXferCount`**
- **`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
- **`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
- **`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

## 63.2 SMARTCARD Firmware driver API description

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

### 63.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a `SMARTCARD_HandleTypeDef` handle structure.
2. Initialize the SMARTCARD low level resources by implementing the `HAL_SMARTCARD_MspInit()` API:
  - a. Enable the interface clock of the USARTx associated to the SMARTCARD.
  - b. SMARTCARD pins configuration:
    - Enable the clock for the SMARTCARD GPIOs.
    - Configure SMARTCARD pins as alternate function pull-up.
  - c. 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.
  - d. 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 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 SMARTCARD DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx stream.
    - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the SMARTCARD Init structure.
4. Initialize the SMARTCARD registers by calling the `HAL_SMARTCARD_Init()` API:
  - These APIs configure 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_ENABLE`: Enable the SMARTCARD peripheral
- `__HAL_SMARTCARD_DISABLE`: Disable the SMARTCARD peripheral
- `__HAL_SMARTCARD_GET_FLAG` : Check whether the specified SMARTCARD flag is set or not
- `__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

*Note:* You can refer to the SMARTCARD HAL driver header file for more useful macros

### 63.2.2 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.
- `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.
- `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.

### 63.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in Smartcard mode.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- For the Smartcard mode only these parameters can be configured:
  - Baud Rate
  - Word Length => Should be 9 bits (8 bits + parity)
  - Stop Bit
  - Parity: => Should be enabled
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes
  - Prescaler
  - GuardTime
  - NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
  - Word Length = 9 Bits
  - 1.5 Stop Bit
  - Even parity
  - BaudRate = 12096 baud
  - Tx and Rx enabled

Please refer to the ISO 7816-3 specification for more details.

*Note:* *It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.*

The HAL\_SMARTCARD\_Init() function follows the USART SmartCard configuration procedures (details for the procedures are available in reference manual (RM0430 for STM32F4X3xx MCUs and RM0402 for STM32F412xx MCUs RM0383 for STM32F411xC/E MCUs and RM0401 for STM32F410xx MCUs RM0090 for STM32F4X5xx/STM32F4X7xx/STM32F429xx/STM32F439xx MCUs RM0390 for STM32F446xx MCUs and RM0386 for STM32F469xx/STM32F479xx MCUs)).

This section contains the following APIs:

- [HAL\\_SMARTCARD\\_Init\(\)](#)
- [HAL\\_SMARTCARD\\_DeInit\(\)](#)
- [HAL\\_SMARTCARD\\_MspInit\(\)](#)

- [HAL\\_SMARTCARD\\_MspDeInit\(\)](#)
- [HAL\\_SMARTCARD\\_ReInit\(\)](#)

### 63.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

1. 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.
2. 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.
3. 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 APIs 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
4. Blocking mode APIs are :
  - HAL\_SMARTCARD\_Transmit()
  - HAL\_SMARTCARD\_Receive()
5. Non Blocking mode APIs with Interrupt are :
  - HAL\_SMARTCARD\_Transmit\_IT()
  - HAL\_SMARTCARD\_Receive\_IT()
  - HAL\_SMARTCARD\_IRQHandler()
6. Non Blocking mode functions with DMA are :
  - HAL\_SMARTCARD\_Transmit\_DMA()
  - HAL\_SMARTCARD\_Receive\_DMA()
7. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_SMARTCARD\_TxCpltCallback()
  - HAL\_SMARTCARD\_RxCpltCallback()
  - HAL\_SMARTCARD\_ErrorCallback()
8. 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()
9. 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()
10. 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 transmission, 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.



(#) 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, These APIs 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() (#) 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() (#) 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() (#) 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 transmission, 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\(\)](#)

### 63.2.5 Peripheral State and Errors functions

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

- [HAL\\_SMARTCARD\\_GetState\(\)](#) API can be helpful to check in run-time the state of the SmartCard peripheral.
- [HAL\\_SMARTCARD\\_GetError\(\)](#) check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL\\_SMARTCARD\\_GetState\(\)](#)
- [HAL\\_SMARTCARD\\_GetError\(\)](#)

### 63.2.6 Detailed description of functions

#### HAL\_SMARTCARD\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_SMARTCARD\_Init (SMARTCARD\_HandleTypeDef \* hsc)**

##### Function description

Initializes the SmartCard mode according to the specified parameters in the SMARTCARD\_InitTypeDef and create the associated handle.

##### Parameters

- **hsc**: Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

##### Return values

- **HAL**: status

#### HAL\_SMARTCARD\_ReInit

##### Function name

**HAL\_StatusTypeDef HAL\_SMARTCARD\_ReInit (SMARTCARD\_HandleTypeDef \* hsc)**

##### Function description

#### HAL\_SMARTCARD\_DeInit

##### Function name

**HAL\_StatusTypeDef HAL\_SMARTCARD\_DeInit (SMARTCARD\_HandleTypeDef \* hsc)**

##### Function description

Deinitializes the USART SmartCard peripheral.

##### Parameters

- **hsc**: Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

##### Return values

- **HAL**: status

### HAL\_SMARTCARD\_MspInit

#### Function name

**void HAL\_SMARTCARD\_MspInit (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

SMARTCARD MSP Init.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **None:**

### HAL\_SMARTCARD\_MspDeInit

#### Function name

**void HAL\_SMARTCARD\_MspDeInit (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

SMARTCARD MSP DeInit.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **None:**

### HAL\_SMARTCARD\_Transmit

#### Function name

**HAL\_StatusTypeDef HAL\_SMARTCARD\_Transmit (SMARTCARD\_HandleTypeDef \* hsc, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

#### Function description

Send an amount of data in blocking mode.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

#### Return values

- **HAL:** status

### HAL\_SMARTCARD\_Receive

#### Function name

**HAL\_StatusTypeDef HAL\_SMARTCARD\_Receive (SMARTCARD\_HandleTypeDef \* hsc, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

#### Function description

Receive an amount of data in blocking mode.

### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

### Return values

- **HAL:** status

### HAL\_SMARTCARD\_Transmit\_IT

#### Function name

HAL\_StatusTypeDef HAL\_SMARTCARD\_Transmit\_IT (SMARTCARD\_HandleTypeDef \* hsc, uint8\_t \* pData, uint16\_t Size)

#### Function description

Send an amount of data in non blocking mode.

### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

### Return values

- **HAL:** status

### HAL\_SMARTCARD\_Receive\_IT

#### Function name

HAL\_StatusTypeDef HAL\_SMARTCARD\_Receive\_IT (SMARTCARD\_HandleTypeDef \* hsc, uint8\_t \* pData, uint16\_t Size)

#### Function description

Receive an amount of data in non blocking mode.

### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

### Return values

- **HAL:** status

### HAL\_SMARTCARD\_Transmit\_DMA

#### Function name

HAL\_StatusTypeDef HAL\_SMARTCARD\_Transmit\_DMA (SMARTCARD\_HandleTypeDef \* hsc, uint8\_t \* pData, uint16\_t Size)

#### Function description

Send an amount of data in non blocking mode.

### Parameters

- **hsc**: Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for 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 \* hsc, uint8\_t \* pData, uint16\_t Size)**

### Function description

Receive an amount of data in non blocking mode.

### Parameters

- **hsc**: Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be received

### Return values

- **HAL**: status

### Notes

- When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bit.s

### HAL\_SMARTCARD\_Abort

### Function name

**HAL\_StatusTypeDef HAL\_SMARTCARD\_Abort (SMARTCARD\_HandleTypeDef \* hsc)**

### Function description

Abort ongoing transfers (blocking mode).

### Parameters

- **hsc**: SMARTCARD 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 PPP InterruptsDisable 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 \* hsc)**

### Function description

Abort ongoing Transmit transfer (blocking mode).

**Parameters**

- **hsc:** SMARTCARD 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 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 \* hsc)**

**Function description**

Abort ongoing Receive transfer (blocking mode).

**Parameters**

- **hsc:** SMARTCARD 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 PPP InterruptsDisable 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 \* hsc)**

**Function description**

Abort ongoing transfers (Interrupt mode).

**Parameters**

- **hsc:** SMARTCARD 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 PPP InterruptsDisable 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 \* hsc)**

#### Function description

Abort ongoing Transmit transfer (Interrupt mode).

#### Parameters

- **hsc:** SMARTCARD 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 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 \* hsc)**

#### Function description

Abort ongoing Receive transfer (Interrupt mode).

#### Parameters

- **hsc:** SMARTCARD 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 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 \* hsc)**

#### Function description

This function handles SMARTCARD interrupt request.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **None:**

### HAL\_SMARTCARD\_TxCpltCallback

#### Function name

**void HAL\_SMARTCARD\_TxCpltCallback (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

Tx Transfer completed callbacks.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **None:**

### HAL\_SMARTCARD\_RxCpltCallback

#### Function name

**void HAL\_SMARTCARD\_RxCpltCallback (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

Rx Transfer completed callback.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **None:**

### HAL\_SMARTCARD\_ErrorCallback

#### Function name

**void HAL\_SMARTCARD\_ErrorCallback (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

SMARTCARD error callback.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **None:**

### HAL\_SMARTCARD\_AbortCpltCallback

#### Function name

**void HAL\_SMARTCARD\_AbortCpltCallback (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

SMARTCARD Abort Complete callback.

#### Parameters

- **hsc:** SMARTCARD handle.

#### Return values

- **None:**



### HAL\_SMARTCARD\_AbortTransmitCpltCallback

#### Function name

**void HAL\_SMARTCARD\_AbortTransmitCpltCallback (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

SMARTCARD Abort Transmit Complete callback.

#### Parameters

- **hsc:** SMARTCARD handle.

#### Return values

- **None:**

### HAL\_SMARTCARD\_AbortReceiveCpltCallback

#### Function name

**void HAL\_SMARTCARD\_AbortReceiveCpltCallback (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

SMARTCARD Abort Receive Complete callback.

#### Parameters

- **hsc:** SMARTCARD handle.

#### Return values

- **None:**

### HAL\_SMARTCARD\_GetState

#### Function name

**HAL\_SMARTCARD\_StateTypeDef HAL\_SMARTCARD\_GetState (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

Return the SMARTCARD handle state.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for SMARTCARD module.

#### Return values

- **HAL:** state

### HAL\_SMARTCARD\_GetError

#### Function name

**uint32\_t HAL\_SMARTCARD\_GetError (SMARTCARD\_HandleTypeDef \* hsc)**

#### Function description

Return the SMARTCARD error code.

#### Parameters

- **hsc:** Pointer to a SMARTCARD\_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.

#### Return values

- **SMARTCARD:** Error Code

## 63.3 SMARTCARD Firmware driver defines

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

### 63.3.1 SMARTCARD

SMARTCARD

***SMARTCARD Clock Phase***

SMARTCARD\_PHASE\_1EDGE

SMARTCARD\_PHASE\_2EDGE

***SMARTCARD Clock Polarity***

SMARTCARD\_POLARITY\_LOW

SMARTCARD\_POLARITY\_HIGH

***SMARTCARD DMA requests***

SMARTCARD\_DMAREQ\_TX

SMARTCARD\_DMAREQ\_RX

***SMARTCARD Error Code***

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

***SMARTCARD Exported Macros***

**\_\_HAL\_SMARTCARD\_RESET\_HANDLE\_STATE**

**Description:**

- Reset SMARTCARD handle gstate & RxState.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_FLUSH\_DRREGISTER**

**Description:**

- Flush the Smartcard DR register.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_GET\_FLAG**

**Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - 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

**Return value:**

- The: new state of **\_\_FLAG\_\_** (TRUE or FALSE).

### **\_\_HAL\_SMARTCARD\_CLEAR\_FLAG**

**Description:**

- Clear the specified Smartcard pending flags.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD\_FLAG\_TC: Transmission Complete flag.
  - SMARTCARD\_FLAG\_RXNE: Receive data register not empty flag.

**Return value:**

- None

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error) and ORE (Overrun error) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

### **\_\_HAL\_SMARTCARD\_CLEAR\_PEFLAG**

**Description:**

- Clear the SMARTCARD PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_CLEAR\_FEFLAG**

**Description:**

- Clear the SMARTCARD FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_CLEAR\_NEFLAG**

**Description:**

- Clear the SMARTCARD NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_CLEAR\_OREFLAG**

**Description:**

- Clear the SMARTCARD ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_CLEAR\_IDLEFLAG**

**Description:**

- Clear the SMARTCARD IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### \_\_HAL\_SMARTCARD\_ENABLE\_IT

**Description:**

- Enable the specified SmartCard interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- \_\_INTERRUPT\_\_: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - 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)

**Return value:**

- None

### \_\_HAL\_SMARTCARD\_DISABLE\_IT

**Description:**

- Disable the specified SmartCard interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- \_\_INTERRUPT\_\_: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - 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)

**Return value:**

- None

### \_\_HAL\_SMARTCARD\_GET\_IT\_SOURCE

**Description:**

- Checks whether the specified SmartCard interrupt has occurred or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SmartCard Handle.
- \_\_IT\_\_: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt

**Return value:**

- The: new state of \_\_IT\_\_ (TRUE or FALSE).

### **\_\_HAL\_SMARTCARD\_ONE\_BIT\_SAMPLE\_ENABLE**

**Description:**

- Macro to enable the SMARTCARD's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle.

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_ONE\_BIT\_SAMPLE\_DISABLE**

**Description:**

- Macro to disable the SMARTCARD's 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. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_DISABLE**

**Description:**

- Disable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_SMARTCARD\_DMA\_REQUEST\_ENABLE**

**Description:**

- Macros to enable the SmartCard DMA request.

**Parameters:**

- `__HANDLE__`: specifies the SmartCard Handle.
- `__REQUEST__`: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
  - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

**Return value:**

- None

## **\_\_HAL\_SMARTCARD\_DMA\_REQUEST\_DISABLE**

**Description:**

- Macros to disable the SmartCard DMA request.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SmartCard Handle.
- **\_\_REQUEST\_\_**: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - SMARTCARD\_DMAREQ\_TX: SmartCard DMA transmit request
  - SMARTCARD\_DMAREQ\_RX: SmartCard DMA receive request

**Return value:**

- None

**SMARTCARD Last Bit**

## **SMARTCARD\_LASTBIT\_DISABLE**

## **SMARTCARD\_LASTBIT\_ENABLE**

**SMARTCARD Mode**

## **SMARTCARD\_MODE\_RX**

## **SMARTCARD\_MODE\_TX**

## **SMARTCARD\_MODE\_TX\_RX**

**SMARTCARD NACK State**

## **SMARTCARD\_NACK\_ENABLE**

## **SMARTCARD\_NACK\_DISABLE**

**SMARTCARD Parity**

## **SMARTCARD\_PARITY\_EVEN**

## **SMARTCARD\_PARITY\_ODD**

**SMARTCARD Prescaler**

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV2**

SYSCLK divided by 2

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV4**

SYSCLK divided by 4

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV6**

SYSCLK divided by 6

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV8**

SYSCLK divided by 8

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV10**

SYSCLK divided by 10

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV12**

SYSCLK divided by 12

## **SMARTCARD\_PRESCALER\_SYSCLK\_DIV14**

SYSCLK divided by 14

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV16**

SYSCLK divided by 16

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV18**

SYSCLK divided by 18

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV20**

SYSCLK divided by 20

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV22**

SYSCLK divided by 22

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV24**

SYSCLK divided by 24

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV26**

SYSCLK divided by 26

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV28**

SYSCLK divided by 28

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV30**

SYSCLK divided by 30

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV32**

SYSCLK divided by 32

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV34**

SYSCLK divided by 34

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV36**

SYSCLK divided by 36

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV38**

SYSCLK divided by 38

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV40**

SYSCLK divided by 40

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV42**

SYSCLK divided by 42

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV44**

SYSCLK divided by 44

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV46**

SYSCLK divided by 46

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV48**

SYSCLK divided by 48

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV50**

SYSCLK divided by 50

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV52**

SYSCLK divided by 52



**SMARTCARD\_PRESCALER\_SYSCLK\_DIV54**

SYSCLK divided by 54

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV56**

SYSCLK divided by 56

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV58**

SYSCLK divided by 58

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV60**

SYSCLK divided by 60

**SMARTCARD\_PRESCALER\_SYSCLK\_DIV62**

SYSCLK divided by 62

***SMARTCARD Number of Stop Bits***

**SMARTCARD\_STOPBITS\_0\_5**

**SMARTCARD\_STOPBITS\_1\_5**

***SMARTCARD Word Length***

**SMARTCARD\_WORDLENGTH\_9B**

## 64 HAL SMBUS Generic Driver

### 64.1 SMBUS Firmware driver registers structures

#### 64.1.1 SMBUS\_InitTypeDef

*SMBUS\_InitTypeDef* is defined in the `stm32f4xx_hal_smbus.h`

##### Data Fields

- *uint32\_t* *ClockSpeed*
- *uint32\_t* *AnalogFilter*
- *uint32\_t* *OwnAddress1*
- *uint32\_t* *AddressingMode*
- *uint32\_t* *DualAddressMode*
- *uint32\_t* *OwnAddress2*
- *uint32\_t* *GeneralCallMode*
- *uint32\_t* *NoStretchMode*
- *uint32\_t* *PacketErrorCheckMode*
- *uint32\_t* *PeripheralMode*

##### Field Documentation

- *uint32\_t* *SMBUS\_InitTypeDef::ClockSpeed*  
Specifies the clock frequency. This parameter must be set to a value lower than 100kHz
- *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 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::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 Periphral is selected. This parameter can be a value of [SMBUS\\_peripheral\\_mode](#)

#### 64.1.2 \_\_SMBUS\_HandleTypeDef

*\_\_SMBUS\_HandleTypeDef* is defined in the `stm32f4xx_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 HAL\_SMBUS\_StateTypeDef State*
- *\_\_IO HAL\_SMBUS\_ModeTypeDef Mode*
- *\_\_IO uint32\_t ErrorCode*
- *\_\_IO uint32\_t Devaddress*
- *\_\_IO uint32\_t EventCount*
- *uint8\_t XferPEC*

#### 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 this parameter can be a value of SMBUS\_OPTIONS
- ***\_\_IO uint32\_t \_\_SMBUS\_HandleTypeDef::PreviousState***  
SMBUS communication Previous state and mode context for internal usage
- ***HAL\_LockTypeDef \_\_SMBUS\_HandleTypeDef::Lock***  
SMBUS locking object
- ***\_\_IO HAL\_SMBUS\_StateTypeDef \_\_SMBUS\_HandleTypeDef::State***  
SMBUS communication state
- ***\_\_IO HAL\_SMBUS\_ModeTypeDef \_\_SMBUS\_HandleTypeDef::Mode***  
SMBUS communication mode
- ***\_\_IO uint32\_t \_\_SMBUS\_HandleTypeDef::ErrorCode***  
SMBUS Error code
- ***\_\_IO uint32\_t \_\_SMBUS\_HandleTypeDef::Devaddress***  
SMBUS Target device address
- ***\_\_IO uint32\_t \_\_SMBUS\_HandleTypeDef::EventCount***  
SMBUS Event counter
- ***uint8\_t \_\_SMBUS\_HandleTypeDef::XferPEC***  
SMBUS PEC data in reception mode

## 64.2 SMBUS Firmware driver API description

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

### 64.2.1 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 Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, General call and Nostretch mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the @ref HAL\_SMBUS\_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...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()
  - End of abort process, @ref HAL\_SMBUS\_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL\_SMBUS\_AbortCpltCallback()
- 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() and @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 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 : Checks 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

*Note:* You can refer to the SMBUS HAL driver header file for more useful macros

### 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\_RegisterXXXCallback() 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.
- 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\_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.
- 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\_SMBUS\_UnRegisterAddrCallback().

By default, after the @ref HAL\_SMBUS\_Init() and when the state is @ref HAL\_SMBUS\_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\_SMBUS\_STATE\_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in @ref HAL\_SMBUS\_STATE\_READY or @ref HAL\_SMBUS\_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.

### 64.2.2 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:
  - Communication Speed
  - Addressing mode
  - Own Address 1
  - Dual Addressing mode
  - Own Address 2
  - 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.

This section contains the following APIs:

- [\*HAL\\_SMBUS\\_Init\(\)\*](#)
- [\*HAL\\_SMBUS\\_DeInit\(\)\*](#)
- [\*HAL\\_SMBUS\\_MspInit\(\)\*](#)
- [\*HAL\\_SMBUS\\_MspDeInit\(\)\*](#)
- [\*HAL\\_SMBUS\\_ConfigAnalogFilter\(\)\*](#)
- [\*HAL\\_SMBUS\\_ConfigDigitalFilter\(\)\*](#)

### 64.2.3 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_Master_Abort_IT()`
  - `HAL_SMBUS_Slave_Transmit_IT()`
  - `HAL_SMBUS_Slave_Receive_IT()`
  - `HAL_SMBUS_EnableAlert_IT()`
  - `HAL_SMBUS_DisableAlert_IT()`

4. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_SMBUS\_MasterTxCpltCallback()
  - HAL\_SMBUS\_MasterRxCpltCallback()
  - HAL\_SMBUS\_SlaveTxCpltCallback()
  - HAL\_SMBUS\_SlaveRxCpltCallback()
  - HAL\_SMBUS\_AddrCallback()
  - HAL\_SMBUS\_ListenCpltCallback()
  - HAL\_SMBUS\_ErrorCallback()
  - HAL\_SMBUS\_AbortCpltCallback()

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()*
- *HAL\_SMBUS\_EV\_IRQHandler()*
- *HAL\_SMBUS\_ER\_IRQHandler()*
- *HAL\_SMBUS\_MasterTxCpltCallback()*
- *HAL\_SMBUS\_MasterRxCpltCallback()*
- *HAL\_SMBUS\_SlaveTxCpltCallback()*
- *HAL\_SMBUS\_SlaveRxCpltCallback()*
- *HAL\_SMBUS\_AddrCallback()*
- *HAL\_SMBUS\_ListenCpltCallback()*
- *HAL\_SMBUS\_ErrorCallback()*
- *HAL\_SMBUS\_AbortCpltCallback()*

#### 64.2.4 Peripheral State, Mode and Error 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\_GetMode()*
- *HAL\_SMBUS\_GetError()*

#### 64.2.5 Detailed description of functions

##### HAL\_SMBUS\_Init

###### Function name

**HAL\_StatusTypeDef HAL\_SMBUS\_Init (SMBUS\_HandleTypeDef \* hsmbus)**

###### Function description

Initializes 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

DeInitializes 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\_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

Transmits in master mode an amount of data in blocking mode.

### 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

### 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

Enable the Address listen 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\_ConfigAnalogFilter

### Function name

HAL\_StatusTypeDef HAL\_SMBUS\_ConfigAnalogFilter (SMBUS\_HandleTypeDef \* hsmbus, uint32\_t AnalogFilter)

### Function description

Configures SMBUS Analog noise filter.

### Parameters

- **hsmbus:** pointer to a SMBUS\_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
- **AnalogFilter:** new state of the Analog filter.

### Return values

- **HAL:** status

### HAL\_SMBUS\_ConfigDigitalFilter

### Function name

HAL\_StatusTypeDef HAL\_SMBUS\_ConfigDigitalFilter (SMBUS\_HandleTypeDef \* hsmbus, uint32\_t DigitalFilter)

### Function description

Configures SMBUS Digital noise filter.

### Parameters

- **hsmbus:** pointer to a SMBUS\_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between 0x00 and 0x0F.

### Return values

- **HAL:** status

### HAL\_SMBUS\_EV\_IRQHandler

### Function name

void HAL\_SMBUS\_EV\_IRQHandler (SMBUS\_HandleTypeDef \* hsmbus)

### Function description

This function handles 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

This function handles 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), value of SMBUS XferOptions definition
- **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\_AbortCpltCallback

#### Function name

**void HAL\_SMBUS\_AbortCpltCallback (SMBUS\_HandleTypeDef \* hsmbus)**

#### Function description

SMBUS abort 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

**HAL\_SMBUS\_StateTypeDef 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\_GetMode

#### Function name

**HAL\_SMBUS\_ModeTypeDef HAL\_SMBUS\_GetMode (SMBUS\_HandleTypeDef \* hsmbus)**

#### Function description

Return the SMBUS Master, Slave or no mode.

#### Parameters

- **hsmbus:** Pointer to a SMBUS\_HandleTypeDef structure that contains the configuration information for SMBUS module

#### Return values

- **HAL:** mode

### 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

## 64.3 SMBUS Firmware driver defines

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

### 64.3.1 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*

**HAL\_SMBUS\_ERROR\_NONE**

No error

**HAL\_SMBUS\_ERROR\_BERR**

BERR error

**HAL\_SMBUS\_ERROR\_ARLO**

ARLO error

**HAL\_SMBUS\_ERROR\_AF**

AF error

**HAL\_SMBUS\_ERROR\_OVR**

OVR error

**HAL\_SMBUS\_ERROR\_TIMEOUT**

Timeout Error

**HAL\_SMBUS\_ERROR\_ALERT**

Alert error

**HAL\_SMBUS\_ERROR\_PECERR**

PEC error

***SMBUS Exported Macros***



### **\_\_HAL\_SMBUS\_RESET\_HANDLE\_STATE**

**Description:**

- Reset SMBUS handle state.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.

**Return value:**

- None

### **\_\_HAL\_SMBUS\_ENABLE\_IT**

**Description:**

- Enable or disable the specified SMBUS interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.
- **\_\_INTERRUPT\_\_**: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SMBUS\_IT\_BUF: Buffer interrupt enable
  - SMBUS\_IT\_EVT: Event interrupt enable
  - SMBUS\_IT\_ERR: Error interrupt enable

**Return value:**

- None

### **\_\_HAL\_SMBUS\_DISABLE\_IT**

### **\_\_HAL\_SMBUS\_GET\_IT\_SOURCE**

**Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.
- **\_\_INTERRUPT\_\_**: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS\_IT\_BUF: Buffer interrupt enable
  - SMBUS\_IT\_EVT: Event interrupt enable
  - SMBUS\_IT\_ERR: Error interrupt enable

**Return value:**

- The: new state of **\_\_INTERRUPT\_\_** (TRUE or FALSE).

### **\_\_HAL\_SMBUS\_GET\_FLAG**

**Description:**

- Checks whether the specified SMBUS flag is set or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - **SMBUS\_FLAG\_SMBALERT**: SMBus Alert flag
  - **SMBUS\_FLAG\_TIMEOUT**: Timeout or Tlow error flag
  - **SMBUS\_FLAG\_PECERR**: PEC error in reception flag
  - **SMBUS\_FLAG\_OVR**: Overrun/Underrun flag
  - **SMBUS\_FLAG\_AF**: Acknowledge failure flag
  - **SMBUS\_FLAG\_ARLO**: Arbitration lost flag
  - **SMBUS\_FLAG\_BERR**: Bus error flag
  - **SMBUS\_FLAG\_TXE**: Data register empty flag
  - **SMBUS\_FLAG\_RXNE**: Data register not empty flag
  - **SMBUS\_FLAG\_STOPF**: Stop detection flag
  - **SMBUS\_FLAG\_ADD10**: 10-bit header sent flag
  - **SMBUS\_FLAG\_BTFF**: Byte transfer finished flag
  - **SMBUS\_FLAG\_ADDR**: Address sent flag Address matched flag
  - **SMBUS\_FLAG\_SB**: Start bit flag
  - **SMBUS\_FLAG\_DUALF**: Dual flag
  - **SMBUS\_FLAG\_SMBHOST**: SMBus host header
  - **SMBUS\_FLAG\_SMBDEFAULT**: SMBus default header
  - **SMBUS\_FLAG\_GENCALL**: General call header flag
  - **SMBUS\_FLAG\_TRA**: Transmitter/Receiver flag
  - **SMBUS\_FLAG\_BUSY**: Bus busy flag
  - **SMBUS\_FLAG\_MSL**: Master/Slave flag

**Return value:**

- The: new state of **\_\_FLAG\_\_** (TRUE or FALSE).

### **\_\_HAL\_SMBUS\_CLEAR\_FLAG**

**Description:**

- Clears the SMBUS pending flags which are cleared by writing 0 in a specific bit.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.
- **\_\_FLAG\_\_**: specifies the flag to clear. This parameter can be any combination of the following values:
  - **SMBUS\_FLAG\_SMBALERT**: SMBus Alert flag
  - **SMBUS\_FLAG\_TIMEOUT**: Timeout or Tlow error flag
  - **SMBUS\_FLAG\_PECERR**: PEC error in reception flag
  - **SMBUS\_FLAG\_OVR**: Overrun/Underrun flag (Slave mode)
  - **SMBUS\_FLAG\_AF**: Acknowledge failure flag
  - **SMBUS\_FLAG\_ARLO**: Arbitration lost flag (Master mode)
  - **SMBUS\_FLAG\_BERR**: Bus error flag

**Return value:**

- None

### **\_\_HAL\_SMBUS\_CLEAR\_ADDRFLAG**

**Description:**

- Clears the SMBUS ADDR pending flag.

**Parameters:**

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.

**Return value:**

- None

### **\_\_HAL\_SMBUS\_CLEAR\_STOPFLAG**

**Description:**

- Clears the SMBUS STOPF pending flag.

**Parameters:**

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1, 2, or 3 to select the SMBUS peripheral.

**Return value:**

- None

### **\_\_HAL\_SMBUS\_ENABLE**

**Description:**

- Enable the SMBUS peripheral.

**Parameters:**

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.

**Return value:**

- None

### **\_\_HAL\_SMBUS\_DISABLE**

**Description:**

- Disable the SMBUS peripheral.

**Parameters:**

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.

**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\_SMBALERT**

**SMBUS\_FLAG\_TIMEOUT**

**SMBUS\_FLAG\_PECERR**

SMBUS\_FLAG\_OVR

SMBUS\_FLAG\_AF

SMBUS\_FLAG\_ARLO

SMBUS\_FLAG\_BERR

SMBUS\_FLAG\_TXE

SMBUS\_FLAG\_RXNE

SMBUS\_FLAG\_STOPF

SMBUS\_FLAG\_ADD10

SMBUS\_FLAG\_BTF

SMBUS\_FLAG\_ADDR

SMBUS\_FLAG\_SB

SMBUS\_FLAG\_DUALF

SMBUS\_FLAG\_SMBHOST

SMBUS\_FLAG\_SMBDEFAULT

SMBUS\_FLAG\_GENCALL

SMBUS\_FLAG\_TRA

SMBUS\_FLAG\_BUSY

SMBUS\_FLAG\_MSL

***SMBUS general call addressing mode***

SMBUS\_GENERALCALL\_DISABLE

SMBUS\_GENERALCALL\_ENABLE

***SMBUS Interrupt configuration definition***

SMBUS\_IT\_BUF

SMBUS\_IT\_EVT

SMBUS\_IT\_ERR

***SMBUS nostretch mode***

SMBUS\_NOSTRETCH\_DISABLE

SMBUS\_NOSTRETCH\_ENABLE

***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 XferDirection definition*

SMBUS\_DIRECTION\_RECEIVE

SMBUS\_DIRECTION\_TRANSMIT

*SMBUS XferOptions definition*

SMBUS\_FIRST\_FRAME

SMBUS\_NEXT\_FRAME

SMBUS\_FIRST\_AND\_LAST\_FRAME\_NO\_PEC

SMBUS\_LAST\_FRAME\_NO\_PEC

SMBUS\_FIRST\_AND\_LAST\_FRAME\_WITH\_PEC

SMBUS\_LAST\_FRAME\_WITH\_PEC

## 65 HAL SPDIFRX Generic Driver

### 65.1 SPDIFRX Firmware driver registers structures

#### 65.1.1 SPDIFRX\_InitTypeDef

*SPDIFRX\_InitTypeDef* is defined in the `stm32f4xx_hal_spdifrx.h`

Data Fields

- *uint32\_t* *InputSelection*
- *uint32\_t* *Retries*
- *uint32\_t* *WaitForActivity*
- *uint32\_t* *ChannelSelection*
- *uint32\_t* *DataFormat*
- *uint32\_t* *StereoMode*
- *uint32\_t* *PreambleTypeMask*
- *uint32\_t* *ChannelStatusMask*
- *uint32\_t* *ValidityBitMask*
- *uint32\_t* *ParityErrorMask*

Field Documentation

- *uint32\_t SPDIFRX\_InitTypeDef::InputSelection*  
Specifies the SPDIF input selection. This parameter can be a value of [SPDIFRX\\_Input\\_Selection](#)
- *uint32\_t SPDIFRX\_InitTypeDef::Retries*  
Specifies the Maximum allowed re-tries during synchronization phase. This parameter can be a value of [SPDIFRX\\_Max\\_Retries](#)
- *uint32\_t SPDIFRX\_InitTypeDef::WaitForActivity*  
Specifies the wait for activity on SPDIF selected input. This parameter can be a value of [SPDIFRX\\_Wait\\_For\\_Activity](#).
- *uint32\_t SPDIFRX\_InitTypeDef::ChannelSelection*  
Specifies whether the control flow will take the channel status from channel A or B. This parameter can be a value of [SPDIFRX\\_Channel\\_Selection](#)
- *uint32\_t SPDIFRX\_InitTypeDef::DataFormat*  
Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of [SPDIFRX\\_Data\\_Format](#)
- *uint32\_t SPDIFRX\_InitTypeDef::StereoMode*  
Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of [SPDIFRX\\_Stereo\\_Mode](#)
- *uint32\_t SPDIFRX\_InitTypeDef::PreambleTypeMask*  
Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PT\\_Mask](#)
- *uint32\_t SPDIFRX\_InitTypeDef::ChannelStatusMask*  
Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_ChannelStatus\\_Mask](#)
- *uint32\_t SPDIFRX\_InitTypeDef::ValidityBitMask*  
Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_V\\_Mask](#)
- *uint32\_t SPDIFRX\_InitTypeDef::ParityErrorMask*  
Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PE\\_Mask](#)

#### 65.1.2 SPDIFRX\_SetDataFormatTypeDef

*SPDIFRX\_SetDataFormatTypeDef* is defined in the `stm32f4xx_hal_spdifrx.h`

**Data Fields**

- *uint32\_t DataFormat*
- *uint32\_t StereoMode*
- *uint32\_t PreambleTypeMask*
- *uint32\_t ChannelStatusMask*
- *uint32\_t ValidityBitMask*
- *uint32\_t ParityErrorMask*

**Field Documentation**

- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::DataFormat*  
Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of [SPDIFRX\\_Data\\_Format](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::StereoMode*  
Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of [SPDIFRX\\_Stereo\\_Mode](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::PreambleTypeMask*  
Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PT\\_Mask](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::ChannelStatusMask*  
Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_ChannelStatus\\_Mask](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::ValidityBitMask*  
Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_V\\_Mask](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::ParityErrorMask*  
Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PE\\_Mask](#)

**65.1.3**
**SPDIFRX\_HandleTypeDef**

*SPDIFRX\_HandleTypeDef* is defined in the stm32f4xx\_hal\_spdifrx.h

**Data Fields**

- *SPDIFRX\_TypeDef \* Instance*
- *SPDIFRX\_InitTypeDef Init*
- *uint32\_t \* pRxBuffPtr*
- *uint32\_t \* pCsBuffPtr*
- *\_\_IO uint16\_t RxXferSize*
- *\_\_IO uint16\_t RxXferCount*
- *\_\_IO uint16\_t CsXferSize*
- *\_\_IO uint16\_t CsXferCount*
- *DMA\_HandleTypeDef \* hdmaCsRx*
- *DMA\_HandleTypeDef \* hdmaDrRx*
- *\_\_IO HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_SPDIFRX\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*

**Field Documentation**

- *SPDIFRX\_TypeDef\* SPDIFRX\_HandleTypeDef::Instance*
- *SPDIFRX\_InitTypeDef SPDIFRX\_HandleTypeDef::Init*
- *uint32\_t\* SPDIFRX\_HandleTypeDef::pRxBuffPtr*
- *uint32\_t\* SPDIFRX\_HandleTypeDef::pCsBuffPtr*
- *\_\_IO uint16\_t SPDIFRX\_HandleTypeDef::RxXferSize*
- *\_\_IO uint16\_t SPDIFRX\_HandleTypeDef::RxXferCount*

- `__IO uint16_t SPDIFRX_HandleTypeDef::CsXferSize`
- `__IO uint16_t SPDIFRX_HandleTypeDef::CsXferCount`
- `DMA_HandleTypeDef* SPDIFRX_HandleTypeDef::hdmaCsRx`
- `DMA_HandleTypeDef* SPDIFRX_HandleTypeDef::hdmaDrRx`
- `__IO HAL_LockTypeDef SPDIFRX_HandleTypeDef::Lock`
- `__IO HAL_SPDIFRX_StateTypeDef SPDIFRX_HandleTypeDef::State`
- `__IO uint32_t SPDIFRX_HandleTypeDef::ErrorCode`

## 65.2 SPDIFRX Firmware driver API description

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

### 65.2.1 How to use this driver

The SPDIFRX HAL driver can be used as follow:

1. Declare SPDIFRX\_HandleTypeDef handle structure.
2. Initialize the SPDIFRX low level resources by implement the HAL\_SPDIFRX\_MspInit() API:
  - a. Enable the SPDIFRX interface clock.
  - b. SPDIFRX pins configuration:
    - Enable the clock for the SPDIFRX GPIOs.
    - Configure these SPDIFRX pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_SPDIFRX\_ReceiveControlFlow\_IT() and HAL\_SPDIFRX\_ReceiveDataFlow\_IT() API's).
    - Configure the SPDIFRX interrupt priority.
    - Enable the NVIC SPDIFRX IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_SPDIFRX\_ReceiveDataFlow\_DMA() and HAL\_SPDIFRX\_ReceiveControlFlow\_DMA() API's).
    - Declare a DMA handle structure for the reception of the Data Flow channel.
    - Declare a DMA handle structure for the reception of the Control Flow channel.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure CtrIRx/DataRx with the required parameters.
    - Configure the DMA Channel.
    - Associate the initialized DMA handle to the SPDIFRX DMA CtrIRx/DataRx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CtrIRx/DataRx channel.
3. Program the input selection, re-tries number, wait for activity, channel status selection, data format, stereo mode and masking of user bits using HAL\_SPDIFRX\_Init() function.

*Note:* The specific SPDIFRX interrupts (RXNE/CSRNE and Error Interrupts) will be managed using the macros `__SPDIFRX_ENABLE_IT()` and `__SPDIFRX_DISABLE_IT()` inside the receive process.

*Note:* Make sure that `ck_spdif` clock is configured.

4. Three operation modes are available within this driver :

#### Polling mode for reception operation (for debug purpose)

- Receive data flow in blocking mode using HAL\_SPDIFRX\_ReceiveDataFlow()
- Receive control flow of data in blocking mode using HAL\_SPDIFRX\_ReceiveControlFlow()

#### Interrupt mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode using HAL\_SPDIFRX\_ReceiveDataFlow\_IT()
- Receive an amount of data (Control Flow) in non blocking mode using HAL\_SPDIFRX\_ReceiveControlFlow\_IT()



- At reception end of half transfer HAL\_SPDIFRX\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxHalfCpltCallback
- At reception end of transfer HAL\_SPDIFRX\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxCpltCallback
- In case of transfer Error, HAL\_SPDIFRX\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_ErrorCallback

#### DMA mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode (DMA) using HAL\_SPDIFRX\_ReceiveDataFlow\_DMA()
- Receive an amount of data (Control Flow) in non blocking mode (DMA) using HAL\_SPDIFRX\_ReceiveControlFlow\_DMA()
- At reception end of half transfer HAL\_SPDIFRX\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxHalfCpltCallback
- At reception end of transfer HAL\_SPDIFRX\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxCpltCallback
- In case of transfer Error, HAL\_SPDIFRX\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_ErrorCallback
- Stop the DMA Transfer using HAL\_SPDIFRX\_DMAStop()

#### SPDIFRX HAL driver macros list

Below the list of most used macros in SPDIFRX HAL driver.

- `__HAL_SPDIFRX_IDLE`: Disable the specified SPDIFRX peripheral (IDEL State)
- `__HAL_SPDIFRX_SYNC`: Enable the synchronization state of the specified SPDIFRX peripheral (SYNC State)
- `__HAL_SPDIFRX_RCV`: Enable the receive state of the specified SPDIFRX peripheral (RCV State)
- `__HAL_SPDIFRX_ENABLE_IT` : Enable the specified SPDIFRX interrupts
- `__HAL_SPDIFRX_DISABLE_IT` : Disable the specified SPDIFRX interrupts
- `__HAL_SPDIFRX_GET_FLAG`: Check whether the specified SPDIFRX flag is set or not.

*Note:* You can refer to the SPDIFRX HAL driver header file for more useful macros

#### Callback registration

### 65.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPDIFRX peripheral:

- User must Implement HAL\_SPDIFRX\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_SPDIFRX\_Init() to configure the SPDIFRX peripheral with the selected configuration:
  - Input Selection (IN0, IN1,...)
  - Maximum allowed re-tries during synchronization phase
  - Wait for activity on SPDIF selected input
  - Channel status selection (from channel A or B)
  - Data format (LSB, MSB, ...)
  - Stereo mode
  - User bits masking (PT,C,U,V,...)
- Call the function HAL\_SPDIFRX\_DeInit() to restore the default configuration of the selected SPDIFRXx peripheral.

This section contains the following APIs:

- [\*\*HAL\\_SPDIFRX\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SPDIFRX\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_SPDIFRX\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_SPDIFRX\\_MspDeInit\(\)\*\*](#)

- [\*HAL\\_SPDIFRX\\_SetDataFormat\(\)\*](#)

### 65.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPDIFRX data transfers.

1. There is two mode 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 start-up. The end of the data processing will be indicated through the dedicated SPDIFRX IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
  - [\*HAL\\_SPDIFRX\\_ReceiveDataFlow\(\)\*](#)
  - [\*HAL\\_SPDIFRX\\_ReceiveControlFlow\(\) \(+@\)\*](#) Do not use blocking mode to receive both control and data flow at the same time.
3. No-Blocking mode functions with Interrupt are :
  - [\*HAL\\_SPDIFRX\\_ReceiveControlFlow\\_IT\(\)\*](#)
  - [\*HAL\\_SPDIFRX\\_ReceiveDataFlow\\_IT\(\)\*](#)
4. No-Blocking mode functions with DMA are :
  - [\*HAL\\_SPDIFRX\\_ReceiveControlFlow\\_DMA\(\)\*](#)
  - [\*HAL\\_SPDIFRX\\_ReceiveDataFlow\\_DMA\(\)\*](#)
5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - [\*HAL\\_SPDIFRX\\_RxCpltCallback\(\)\*](#)
  - [\*HAL\\_SPDIFRX\\_CxCpltCallback\(\)\*](#)

This section contains the following APIs:

- [\*HAL\\_SPDIFRX\\_ReceiveDataFlow\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_ReceiveControlFlow\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_ReceiveDataFlow\\_IT\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_ReceiveControlFlow\\_IT\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_ReceiveDataFlow\\_DMA\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_ReceiveControlFlow\\_DMA\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_DMAStop\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_CxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_CxCpltCallback\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_ErrorCallback\(\)\*](#)

### 65.2.4 Peripheral State and Errors 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\\_SPDIFRX\\_GetState\(\)\*](#)
- [\*HAL\\_SPDIFRX\\_GetError\(\)\*](#)

### 65.2.5 Detailed description of functions

#### HAL\_SPDIFRX\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_Init (SPDIFRX\_HandleTypeDef \* hspdif)**

### Function description

Initializes the SPDIFRX according to the specified parameters in the SPDIFRX\_InitTypeDef and create the associated handle.

### Parameters

- **hspdif**: SPDIFRX handle

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_DeInit

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_DeInit (SPDIFRX\_HandleTypeDef \* hspdif)**

### Function description

Deinitializes the SPDIFRX peripheral.

### Parameters

- **hspdif**: SPDIFRX handle

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_MspInit

### Function name

**void HAL\_SPDIFRX\_MspInit (SPDIFRX\_HandleTypeDef \* hspdif)**

### Function description

SPDIFRX MSP Init.

### Parameters

- **hspdif**: SPDIFRX handle

### Return values

- **None**:

### HAL\_SPDIFRX\_MspDeInit

### Function name

**void HAL\_SPDIFRX\_MspDeInit (SPDIFRX\_HandleTypeDef \* hspdif)**

### Function description

SPDIFRX MSP DeInit.

### Parameters

- **hspdif**: SPDIFRX handle

### Return values

- **None**:

### HAL\_SPDIFRX\_SetDataFormat

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_SetDataFormat (SPDIFRX\_HandleTypeDef \* hspdif, SPDIFRX\_SetDataFormatTypeDef sDataFormat)**

### Function description

Set the SPDIFRX data format according to the specified parameters in the SPDIFRX\_InitTypeDef.

### Parameters

- **hspdif**: SPDIFRX handle
- **sDataFormat**: SPDIFRX data format

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_ReceiveDataFlow

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_ReceiveDataFlow (SPDIFRX\_HandleTypeDef \* hspdif, uint32\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

### Function description

Receives an amount of data (Data Flow) in blocking mode.

### Parameters

- **hspdif**: pointer to SPDIFRX\_HandleTypeDef structure that contains the configuration information for SPDIFRX module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be received
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_ReceiveControlFlow

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_ReceiveControlFlow (SPDIFRX\_HandleTypeDef \* hspdif, uint32\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

### Function description

Receives an amount of data (Control Flow) in blocking mode.

### Parameters

- **hspdif**: pointer to a SPDIFRX\_HandleTypeDef structure that contains the configuration information for SPDIFRX module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be received
- **Timeout**: Timeout duration

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_ReceiveControlFlow\_IT

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_ReceiveControlFlow\_IT (SPDIFRX\_HandleTypeDef \* hspdif, uint32\_t \* pData, uint16\_t Size)**

### Function description

Receive an amount of data (Control Flow) with Interrupt.

### Parameters

- **hspdif**: SPDIFRX handle
- **pData**: a 32-bit pointer to the Receive data buffer.
- **Size**: number of data sample (Control Flow) to be received

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_ReceiveDataFlow\_IT

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_ReceiveDataFlow\_IT (SPDIFRX\_HandleTypeDef \* hspdif, uint32\_t \* pData, uint16\_t Size)**

### Function description

Receive an amount of data (Data Flow) in non-blocking mode with Interrupt.

### Parameters

- **hspdif**: SPDIFRX handle
- **pData**: a 32-bit pointer to the Receive data buffer.
- **Size**: number of data sample to be received .

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_IRQHandler

### Function name

**void HAL\_SPDIFRX\_IRQHandler (SPDIFRX\_HandleTypeDef \* hspdif)**

### Function description

This function handles SPDIFRX interrupt request.

### Parameters

- **hspdif**: SPDIFRX handle

### Return values

- **HAL**: status

### HAL\_SPDIFRX\_ReceiveControlFlow\_DMA

### Function name

**HAL\_StatusTypeDef HAL\_SPDIFRX\_ReceiveControlFlow\_DMA (SPDIFRX\_HandleTypeDef \* hspdif, uint32\_t \* pData, uint16\_t Size)**

### Function description

Receive an amount of data (Control Flow) with DMA.

### Parameters

- **hspdif**: SPDIFRX handle
- **pData**: a 32-bit pointer to the Receive data buffer.
- **Size**: number of data (Control Flow) sample to be received

### Return values

- **HAL**: status

## HAL\_SPDIFRX\_ReceiveDataFlow\_DMA

### Function name

HAL\_StatusTypeDef HAL\_SPDIFRX\_ReceiveDataFlow\_DMA (SPDIFRX\_HandleTypeDef \* hspdif, uint32\_t \* pData, uint16\_t Size)

### Function description

Receive an amount of data (Data Flow) mode with DMA.

### Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be received

### Return values

- **HAL:** status

## HAL\_SPDIFRX\_DMAStop

### Function name

HAL\_StatusTypeDef HAL\_SPDIFRX\_DMAStop (SPDIFRX\_HandleTypeDef \* hspdif)

### Function description

stop the audio stream receive from the Media.

### Parameters

- **hspdif:** SPDIFRX handle

### Return values

- **None:**

## HAL\_SPDIFRX\_RxHalfCpltCallback

### Function name

void HAL\_SPDIFRX\_RxHalfCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)

### Function description

Rx Transfer (Data flow) half completed callbacks.

### Parameters

- **hspdif:** SPDIFRX handle

### Return values

- **None:**

## HAL\_SPDIFRX\_RxCpltCallback

### Function name

void HAL\_SPDIFRX\_RxCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)

### Function description

Rx Transfer (Data flow) completed callbacks.

### Parameters

- **hspdif:** SPDIFRX handle

**Return values**

- **None:**

**HAL\_SPDIFRX\_ErrorCallback**

**Function name**

**void HAL\_SPDIFRX\_ErrorCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

**Function description**

SPDIFRX error callbacks.

**Parameters**

- **hspdif:** SPDIFRX handle

**Return values**

- **None:**

**HAL\_SPDIFRX\_CxHalfCpltCallback**

**Function name**

**void HAL\_SPDIFRX\_CxHalfCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

**Function description**

Rx (Control flow) Transfer half completed callbacks.

**Parameters**

- **hspdif:** SPDIFRX handle

**Return values**

- **None:**

**HAL\_SPDIFRX\_CxCpltCallback**

**Function name**

**void HAL\_SPDIFRX\_CxCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

**Function description**

Rx Transfer (Control flow) completed callbacks.

**Parameters**

- **hspdif:** SPDIFRX handle

**Return values**

- **None:**

**HAL\_SPDIFRX\_GetState**

**Function name**

**HAL\_SPDIFRX\_StateTypeDef HAL\_SPDIFRX\_GetState (SPDIFRX\_HandleTypeDef const \*const hspdif)**

**Function description**

Return the SPDIFRX state.

**Parameters**

- **hspdif:** SPDIFRX handle

**Return values**

- **HAL:** state

## HAL\_SPDIFRX\_GetError

### Function name

`uint32_t HAL_SPDIFRX_GetError (SPDIFRX_HandleTypeDef const *const hspdif)`

### Function description

Return the SPDIFRX error code.

### Parameters

- **hspdif**: SPDIFRX handle

### Return values

- **SPDIFRX**: Error Code

## 65.3 SPDIFRX Firmware driver defines

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

### 65.3.1 SPDIFRX

SPDIFRX

*SPDIFRX Channel Status Mask*

SPDIFRX\_CHANNELSTATUS\_OFF

SPDIFRX\_CHANNELSTATUS\_ON

*SPDIFRX Channel Selection*

SPDIFRX\_CHANNEL\_A

SPDIFRX\_CHANNEL\_B

*SPDIFRX Data Format*

SPDIFRX\_DATAFORMAT\_LSB

SPDIFRX\_DATAFORMAT\_MSB

SPDIFRX\_DATAFORMAT\_32BITS

*SPDIFRX Error Code*

HAL\_SPDIFRX\_ERROR\_NONE

No error

HAL\_SPDIFRX\_ERROR\_TIMEOUT

Timeout error

HAL\_SPDIFRX\_ERROR\_OVR

OVR error

HAL\_SPDIFRX\_ERROR\_PE

Parity error

HAL\_SPDIFRX\_ERROR\_DMA

DMA transfer error

HAL\_SPDIFRX\_ERROR\_UNKNOWN

Unknown Error error



### *SPDIFRX Exported Macros*

#### **\_\_HAL\_SPDIFRX\_RESET\_HANDLE\_STATE**

**Description:**

- Reset SPDIFRX handle state.

**Parameters:**

- `__HANDLE__`: SPDIFRX handle.

**Return value:**

- None

#### **\_\_HAL\_SPDIFRX\_IDLE**

**Description:**

- Disable the specified SPDIFRX peripheral (IDLE State).

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.

**Return value:**

- None

#### **\_\_HAL\_SPDIFRX\_SYNC**

**Description:**

- Enable the specified SPDIFRX peripheral (SYNC State).

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.

**Return value:**

- None

#### **\_\_HAL\_SPDIFRX\_RCV**

**Description:**

- Enable the specified SPDIFRX peripheral (RCV State).

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.

**Return value:**

- None

#### **\_\_HAL\_SPDIFRX\_ENABLE\_IT**

**Description:**

- Enable or disable the specified SPDIFRX interrupts.

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPDIFRX\_IT\_RXNE
  - SPDIFRX\_IT\_CSRNE
  - SPDIFRX\_IT\_PERRIE
  - SPDIFRX\_IT\_OVRIE
  - SPDIFRX\_IT\_SBLKIE
  - SPDIFRX\_IT\_SYNCDIE
  - SPDIFRX\_IT\_IFEIE

**Return value:**

- None

## `__HAL_SPDIFRX_DISABLE_IT`

## `__HAL_SPDIFRX_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.
- `__INTERRUPT__`: specifies the SPDIFRX interrupt source to check. This parameter can be one of the following values:
  - `SPDIFRX_IT_RXNE`
  - `SPDIFRX_IT_CSRNE`
  - `SPDIFRX_IT_PERRIE`
  - `SPDIFRX_IT_OVRIE`
  - `SPDIFRX_IT_SBLKIE`
  - `SPDIFRX_IT_SYNCDIE`
  - `SPDIFRX_IT_IFEIE`

**Return value:**

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

## `__HAL_SPDIFRX_GET_FLAG`

**Description:**

- Checks whether the specified SPDIFRX flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `SPDIFRX_FLAG_RXNE`
  - `SPDIFRX_FLAG_CSRNE`
  - `SPDIFRX_FLAG_PERR`
  - `SPDIFRX_FLAG_OVR`
  - `SPDIFRX_FLAG_SBD`
  - `SPDIFRX_FLAG_SYNCD`
  - `SPDIFRX_FLAG_FERR`
  - `SPDIFRX_FLAG_SERR`
  - `SPDIFRX_FLAG_TERR`

**Return value:**

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

## `__HAL_SPDIFRX_CLEAR_IT`

**Description:**

- Clears the specified SPDIFRX SR flag, in setting the proper IFCR register bit.

**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:
  - `SPDIFRX_FLAG_PERR`
  - `SPDIFRX_FLAG_OVR`
  - `SPDIFRX_SR_SBD`
  - `SPDIFRX_SR_SYNCD`

**Return value:**

- None

***SPDIFRX Flags Definition***

SPDIFRX\_FLAG\_RXNE

SPDIFRX\_FLAG\_CSRNE

SPDIFRX\_FLAG\_PERR

SPDIFRX\_FLAG\_OVR

SPDIFRX\_FLAG\_SBD

SPDIFRX\_FLAG\_SYNCDC

SPDIFRX\_FLAG\_FERR

SPDIFRX\_FLAG\_SERR

SPDIFRX\_FLAG\_TERR

***SPDIFRX Input Selection***

SPDIFRX\_INPUT\_IN0

SPDIFRX\_INPUT\_IN1

SPDIFRX\_INPUT\_IN2

SPDIFRX\_INPUT\_IN3

***SPDIFRX Interrupts Definition***

SPDIFRX\_IT\_RXNE

SPDIFRX\_IT\_CSRNE

SPDIFRX\_IT\_PERRIE

SPDIFRX\_IT\_OVRIE

SPDIFRX\_IT\_SBLKIE

SPDIFRX\_IT\_SYNCDCIE

SPDIFRX\_IT\_IFEIE

***SPDIFRX Maximum Retries***

SPDIFRX\_MAXRETRIES\_NONE

SPDIFRX\_MAXRETRIES\_3

SPDIFRX\_MAXRETRIES\_15

SPDIFRX\_MAXRETRIES\_63

***SPDIFRX Parity Error Mask***

SPDIFRX\_PARITYERRORMASK\_OFF

SPDIFRX\_PARITYERRORMASK\_ON

*SPDIFRX Preamble Type Mask*

SPDIFRX\_PREAMBLETYPEMASK\_OFF

SPDIFRX\_PREAMBLETYPEMASK\_ON

*SPDIFRX State*

SPDIFRX\_STATE\_IDLE

SPDIFRX\_STATE\_SYNC

SPDIFRX\_STATE\_RCV

*SPDIFRX Stereo Mode*

SPDIFRX\_STEREOMODE\_DISABLE

SPDIFRX\_STEREOMODE\_ENABLE

*SPDIFRX Validity Mask*

SPDIFRX\_VALIDITYMASK\_OFF

SPDIFRX\_VALIDITYMASK\_ON

*SPDIFRX Wait For Activity*

SPDIFRX\_WAITFORACTIVITY\_OFF

SPDIFRX\_WAITFORACTIVITY\_ON

## 66 HAL SPI Generic Driver

### 66.1 SPI Firmware driver registers structures

#### 66.1.1 SPI\_InitTypeDef

*SPI\_InitTypeDef* is defined in the `stm32f4xx_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*

##### 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`

### 66.1.2 `__SPI_HandleTypeDef`

`__SPI_HandleTypeDef` is defined in the `stm32f4xx_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`
- `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
- `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

## 66.2 SPI Firmware driver API description

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

### 66.2.1 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.

## 66.2.2 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
- 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\\_Msplnit\(\)\*](#)
- [\*HAL\\_SPI\\_MspDeInit\(\)\*](#)

### 66.2.3 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\\_DMAMPause\(\)\*](#)
- [\*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()*

#### 66.2.4 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()*

#### 66.2.5 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

## 66.3 SPI Firmware driver defines

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

### 66.3.1 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 Data Size**

SPI\_DATASIZE\_8BIT

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 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

**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 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\_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 Slave Select Management*

SPI\_NSS\_SOFT

SPI\_NSS\_HARD\_INPUT

SPI\_NSS\_HARD\_OUTPUT

*SPI TI Mode*

SPI\_TIMODE\_DISABLE

SPI\_TIMODE\_ENABLE

## 67 HAL SRAM Generic Driver

### 67.1 SRAM Firmware driver registers structures

#### 67.1.1 SRAM\_HandleTypeDef

*SRAM\_HandleTypeDef* is defined in the `stm32f4xx_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

### 67.2 SRAM Firmware driver API description

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

#### 67.2.1 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/FSMC 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.

## 67.2.2 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\(\)](#)

## 67.2.3 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()`

#### 67.2.4 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()`

#### 67.2.5 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()`

#### 67.2.6 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\_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\_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\_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

## 67.3 SRAM Firmware driver defines

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

### 67.3.1 SRAM

SRAM

#### SRAM Exported Macros

#### `__HAL_SRAM_RESET_HANDLE_STATE`

##### Description:

- Reset SRAM handle state.

##### Parameters:

- `__HANDLE__`: SRAM handle

##### Return value:

- None

## 68 HAL TIM Generic Driver

### 68.1 TIM Firmware driver registers structures

#### 68.1.1 TIM\_Base\_InitTypeDef

*TIM\_Base\_InitTypeDef* is defined in the `stm32f4xx_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`
- *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`.
- *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](#)

#### 68.1.2 TIM\_OC\_InitTypeDef

*TIM\_OC\_InitTypeDef* is defined in the `stm32f4xx_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`
- ***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.

### 68.1.3

#### TIM\_OnePulse\_InitTypeDef

*TIM\_OnePulse\_InitTypeDef* is defined in the `stm32f4xx_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`
- ***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

#### 68.1.4

##### TIM\_IC\_InitTypeDef

*TIM\_IC\_InitTypeDef* is defined in the `stm32f4xx_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

#### 68.1.5

##### TIM\_Encoder\_InitTypeDef

*TIM\_Encoder\_InitTypeDef* is defined in the `stm32f4xx_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

### 68.1.6

#### TIM\_ClockConfigTypeDef

*TIM\_ClockConfigTypeDef* is defined in the stm32f4xx\_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

### 68.1.7

#### TIM\_ClearInputConfigTypeDef

*TIM\_ClearInputConfigTypeDef* is defined in the stm32f4xx\_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

### 68.1.8

#### **TIM\_MasterConfigTypeDef**

**TIM\_MasterConfigTypeDef** is defined in the stm32f4xx\_hal\_tim.h

##### Data Fields

- ***uint32\_t MasterOutputTrigger***
- ***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::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.

### 68.1.9

#### **TIM\_SlaveConfigTypeDef**

**TIM\_SlaveConfigTypeDef** is defined in the stm32f4xx\_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

### 68.1.10

#### **TIM\_BreakDeadTimeConfigTypeDef**

**TIM\_BreakDeadTimeConfigTypeDef** is defined in the stm32f4xx\_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 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::AutomaticOutput***  
TIM Automatic Output Enable state This parameter can be a value of [TIM\\_AOE\\_Bit\\_Set\\_Reset](#)

**68.1.11 TIM\_HandleTypeDef**

*TIM\_HandleTypeDef* is defined in the stm32f4xx\_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 ChannelINState*
- *\_\_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[4]***  
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

## 68.2 TIM Firmware driver API description

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

### 68.2.1 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

### 68.2.2 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 and HAL\_TIM\_OC\_ConfigChannel: 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.

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.

### 68.2.3 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\(\)\*](#)

### 68.2.4 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()*

### 68.2.5 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()*

### 68.2.6 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()*

### 68.2.7 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()*

### 68.2.8 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()*

### 68.2.9 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

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()*

## 68.2.10 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

**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

**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

### 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

#### 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

### 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

### 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 Tlx 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

### 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).

### 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).

### 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
- **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\_18TRANSFERS.

**Return values**

- **HAL:** status

**Notes**

- This function should be used only when BurstLength is equal to DMA data transfer length.

**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
- **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\_18TRANSFERS.

## Return values

- **HAL:** status

## Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

### 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

### 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 are relevant only for timer instances supporting a break input.

### 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 T11 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 output 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
- **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM\_CCx\_ENABLE or TIM\_CCx\_DISABLE.

#### Return values

- **None:**

## 68.3 TIM Firmware driver defines

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

### 68.3.1 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 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 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\_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 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 Tlx clock sources

**TIM\_CLOCKPOLARITY\_FALLING**

Polarity for Tlx clock sources

**TIM\_CLOCKPOLARITY\_BOTHEDGE**

Polarity for Tlx 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 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\_DCR

TIM\_DMABASE\_DMAR

***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 DMA Sources**

#### TIM\_DMA\_UPDATE

DMA request is triggered by the update event

#### TIM\_DMA\_CC1

DMA request is triggered by the capture/compare match 1 event

#### TIM\_DMA\_CC2

DMA request is triggered by the capture/compare match 2 event event

#### TIM\_DMA\_CC3

DMA request is triggered by the capture/compare match 3 event event

#### TIM\_DMA\_CC4

DMA request is triggered by the capture/compare match 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 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 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

**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

**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_COM`: Commutation interrupt flag
  - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
  - `TIM_FLAG_BREAK`: 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

### 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_COM`: Commutation interrupt flag
  - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
  - `TIM_FLAG_BREAK`: 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

### 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

### 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

### Return value:

- None

## \_\_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
- **\_\_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

**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

**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

**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

**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

**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\_COM

Commutation interrupt flag

#### TIM\_FLAG\_TRIGGER

Trigger interrupt flag

#### TIM\_FLAG\_BREAK

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 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 Tlx source

#### TIM\_INPUTCHANNELPOLARITY\_FALLING

Polarity for Tlx source

#### TIM\_INPUTCHANNELPOLARITY\_BOTHEDGE

Polarity for Tlx 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 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 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 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 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\_NONE**

No trigger selected

## 69 HAL TIM Extension Driver

### 69.1 TIMEx Firmware driver registers structures

#### 69.1.1 TIM\_HallSensor\_InitTypeDef

*TIM\_HallSensor\_InitTypeDef* is defined in the `stm32f4xx_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`

### 69.2 TIMEx Firmware driver API description

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

#### 69.2.1 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

#### 69.2.2 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. Activate the TIM peripheral using one of the start functions:
  - Complementary Output Compare : HAL\_TIMEx\_OC\_N\_Start(), HAL\_TIMEx\_OC\_N\_Start\_DMA(), HAL\_TIMEx\_OC\_N\_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()

### 69.2.3 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()*

### 69.2.4 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\_OC\_N\_Start()*
- *HAL\_TIMEx\_OC\_N\_Stop()*
- *HAL\_TIMEx\_OC\_N\_Start\_IT()*
- *HAL\_TIMEx\_OC\_N\_Stop\_IT()*
- *HAL\_TIMEx\_OC\_N\_Start\_DMA()*

- [\*HAL\\_TIMEx\\_OCN\\_Stop\\_DMA\(\)\*](#)

### 69.2.5 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\(\)\*](#)

### 69.2.6 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\(\)\*](#)

### 69.2.7 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.

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\\_RemapConfig\(\)](#)

### 69.2.8 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\(\)](#)

### 69.2.9 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\(\)](#)

### 69.2.10 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

### 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

### 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

### 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

### 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
- **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

### 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

### 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

### 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

### 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

### 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
- **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

**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\_NONE: No trigger is needed
- **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 T11 of the Interface Timer detect a commutation at its input T11.

## 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\_NONE: No trigger is needed
- **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 T11 of the Interface Timer detect a commutation at its input T11.

## 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\_NONE: No trigger is needed
- **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 T11 of the Interface Timer detect a commutation at its input T11.
- 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\_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 have the following values: (\*\*)  
 – TIM\_TIM1\_TIM3\_TRGO: TIM1 ITR2 is connected to TIM3 TRGO  
 – TIM\_TIM1\_LPTIM: TIM1 ITR2 is connected to LPTIM1 output  
 For TIM2, the parameter can have the following values: (\*\*)  
 – TIM\_TIM2\_TIM8\_TRGO: TIM2 ITR1 is connected to TIM8 TRGO (\*)  
 – TIM\_TIM2\_ETH\_PTP: TIM2 ITR1 is connected to PTP trigger output (\*)  
 – TIM\_TIM2\_USBFS\_SOF: TIM2 ITR1 is connected to OTG FS SOF  
 – TIM\_TIM2\_USBHS\_SOF: TIM2 ITR1 is connected to OTG FS SOF  
 For TIM5, the parameter can have the following values:  
 – TIM\_TIM5\_GPIO: TIM5 TI4 is connected to GPIO  
 – TIM\_TIM5\_LSI: TIM5 TI4 is connected to LSI  
 – TIM\_TIM5\_LSE: TIM5 TI4 is connected to LSE  
 – TIM\_TIM5\_RTC: TIM5 TI4 is connected to the RTC wakeup interrupt  
 – TIM\_TIM5\_TIM3\_TRGO: TIM5 ITR1 is connected to TIM3 TRGO (\*)  
 – TIM\_TIM5\_LPTIM: TIM5 ITR1 is connected to LPTIM1 output (\*)  
 For TIM9, the parameter can have the following values: (\*\*)  
 – TIM\_TIM9\_TIM3\_TRGO: TIM9 ITR1 is connected to TIM3 TRGO  
 – TIM\_TIM9\_LPTIM: TIM9 ITR1 is connected to LPTIM1 output  
 For TIM11, the parameter can have the following values:  
 – TIM\_TIM11\_GPIO: TIM11 TI1 is connected to GPIO  
 – TIM\_TIM11\_HSE: TIM11 TI1 is connected to HSE\_RTC clock  
 – TIM\_TIM11\_SPDIFRX: TIM11 TI1 is connected to SPDIFRX\_FRAME\_SYNC (\*)  
 (\*) Value not defined in all devices.  
 (\*\*) Register not available in all devices.

## 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\_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\_GetChannelINState**

**Function name**

**HAL\_TIM\_ChannelStateTypeDef HAL\_TIMEx\_GetChannelINState (TIM\_HandleTypeDef \* htim, uint32\_t ChannelIN)**

**Function description**

Return actual state of the TIM complementary channel.

**Parameters**

- **htim**: TIM handle
- **ChannelIN**: 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

**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**:

## 69.3 TIMEx Firmware driver defines

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

### 69.3.1 TIMEx

TIMEx

***TIM Extended Remapping***

#### TIM\_TIM2\_TIM8\_TRGO

TIM2 ITR1 is connected to TIM8 TRGO

#### TIM\_TIM2\_USBFS\_SOF

TIM2 ITR1 is connected to OTG FS SOF

#### TIM\_TIM2\_USBHS\_SOF

TIM2 ITR1 is connected to OTG HS SOF

#### TIM\_TIM5\_GPIO

TIM5 TI4 is connected to GPIO

#### TIM\_TIM5\_LSI

TIM5 TI4 is connected to LSI

#### TIM\_TIM5\_LSE

TIM5 TI4 is connected to LSE

#### TIM\_TIM5\_RTC

TIM5 TI4 is connected to the RTC wakeup interrupt

#### TIM\_TIM11\_GPIO

TIM11 TI1 is connected to GPIO

#### TIM\_TIM11\_HSE

TIM11 TI1 is connected to HSE\_RTC clock

## 70 HAL UART Generic Driver

### 70.1 UART Firmware driver registers structures

#### 70.1.1 UART\_InitTypeDef

*UART\_InitTypeDef* is defined in the `stm32f4xx_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**

##### Field Documentation

- *uint32\_t* **UART\_InitTypeDef::BaudRate**  
 This member configures the UART communication baud rate. The baud rate is computed using the following formula:
  - $IntegerDivider = ((PCLKx) / (8 * (OVR8+1) * (huart->Init.BaudRate)))$
  - $FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 8 * (OVR8+1)) + 0.5$  Where OVR8 is the "oversampling by 8 mode" configuration bit in the CR1 register.
- *uint32\_t* **UART\_InitTypeDef::WordLength**  
 Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UART\\_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 fPCLK/8). This parameter can be a value of [UART\\_Over\\_Sampling](#)

#### 70.1.2 \_\_UART\_HandleTypeDef

*\_\_UART\_HandleTypeDef* is defined in the `stm32f4xx_hal_uart.h`

##### Data Fields

- *USART\_TypeDef \** **Instance**
- *UART\_InitTypeDef* **Init**
- *uint8\_t \** **pTxBuffPtr**
- *uint16\_t* **TxXferSize**
- *\_\_IO uint16\_t* **TxXferCount**
- *uint8\_t \** **pRxBuffPtr**

- *uint16\_t RxXferSize*
- *\_\_IO uint16\_t RxXferCount*
- *\_\_IO HAL\_UART\_RxTypeTypeDef ReceptionType*
- *DMA\_HandleTypeDef \* hdmatrix*
- *DMA\_HandleTypeDef \* hdmatrix*
- *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
- ***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
- ***\_\_IO HAL\_UART\_RxTypeTypeDef \_\_UART\_HandleTypeDef::ReceptionType***  
Type of ongoing reception
- ***DMA\_HandleTypeDef\* \_\_UART\_HandleTypeDef::hdmatrix***  
UART Tx DMA Handle parameters
- ***DMA\_HandleTypeDef\* \_\_UART\_HandleTypeDef::hdmatrix***  
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

## 70.2 UART Firmware driver API description

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

### 70.2.1 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:
  - a. Enable the USARTx interface clock.
  - b. UART pins configuration:
    - Enable the clock for the UART GPIOs.
    - Configure these UART pins (TX as alternate function pull-up, RX as alternate function Input).
  - c. 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.
  - d. 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 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 UART DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx stream.
    - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the huart Init structure.
4. For the UART asynchronous mode, initialize the UART registers by calling the HAL\_UART\_Init() API.
5. For the UART Half duplex mode, initialize the UART registers by calling the HAL\_HalfDuplex\_Init() API.
6. For the LIN mode, initialize the UART registers by calling the HAL\_LIN\_Init() API.
7. For the Multi-Processor mode, initialize the UART registers by calling the HAL\_MultiProcessor\_Init() API.

*Note:* The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros `__HAL_UART_ENABLE_IT()` and `__HAL_UART_DISABLE_IT()` inside the transmit and receive process.

*Note:* These APIs (HAL\_UART\_Init() and HAL\_HalfDuplex\_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_UART\_MspInit() API.

## 70.2.2 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.
- 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.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit.

For specific callback RxEventCallback, use dedicated registration/reset functions: respectively @ref HAL\_UART\_RegisterRxEventCallback() , @ref HAL\_UART\_UnRegisterRxEventCallback().

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.

Three operation modes are available within this driver :

#### **Polling mode IO operation**

- Send an amount of data in blocking mode using HAL\_UART\_Transmit()
- Receive an amount of data in blocking mode using HAL\_UART\_Receive()

#### **Interrupt mode IO operation**

- Send an amount of data in non blocking mode using HAL\_UART\_Transmit\_IT()
- At transmission end of transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_UART\_Receive\_IT()
- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback

#### **DMA mode IO operation**

- Send an amount of data in non blocking mode (DMA) using HAL\_UART\_Transmit\_DMA()
- At transmission end of half transfer HAL\_UART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_UART\_Receive\_DMA()
- At reception end of half transfer HAL\_UART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxHalfCpltCallback
- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback

- Pause the DMA Transfer using HAL\_UART\_DMABPause()
- Resume the DMA Transfer using HAL\_UART\_DMABResume()
- Stop the DMA Transfer using HAL\_UART\_DMABStop()

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown).

1. Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller : (+) Detection of inactivity period (RX line has not been active for a given period).
  - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
2. There are two mode of transfer: (+) Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer. (+) Non-Blocking mode: The reception 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 UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_UARTEx\_RxEventCallback() user callback will be executed during Receive process The HAL\_UART\_ErrorCallback()user callback will be executed when a reception error is detected.
3. Blocking mode API: (+) HAL\_UARTEx\_ReceiveToldle()
4. Non-Blocking mode API with Interrupt: (+) HAL\_UARTEx\_ReceiveToldle\_IT()
5. Non-Blocking mode API with DMA: (+) HAL\_UARTEx\_ReceiveToldle\_DMA()

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown). (#) Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller :

- Detection of inactivity period (RX line has not been active for a given period).
  - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte. (#) There are two mode of transfer:
- Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer.
- Non-Blocking mode: The reception 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 UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_UARTEx\_RxEventCallback() user callback will be executed during Receive process The HAL\_UART\_ErrorCallback()user callback will be executed when a reception error is detected. (#) Blocking mode API:
- HAL\_UARTEx\_ReceiveToldle() (#) Non-Blocking mode API with Interrupt:
- HAL\_UARTEx\_ReceiveToldle\_IT() (#) Non-Blocking mode API with DMA:
- HAL\_UARTEx\_ReceiveToldle\_DMA()

### UART HAL driver macros list

Below the list of most used macros in UART HAL driver.

- \_\_HAL\_UART\_ENABLE: Enable the UART peripheral
- \_\_HAL\_UART\_DISABLE: Disable the UART peripheral
- \_\_HAL\_UART\_GET\_FLAG : Check whether the specified UART flag is set or not
- \_\_HAL\_UART\_CLEAR\_FLAG : Clear the specified UART pending flag
- \_\_HAL\_UART\_ENABLE\_IT: Enable the specified UART interrupt
- \_\_HAL\_UART\_DISABLE\_IT: Disable the specified UART interrupt
- \_\_HAL\_UART\_GET\_IT\_SOURCE: Check whether the specified UART interrupt has occurred or not

*Note:* You can refer to the UART HAL driver header file for more useful macros



**Note:** *If the parity is enabled, the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. The UART frame format depends on the frame length defined by the M bit (8-bits or 9-bits). For more details, refer to Table Frame formats in Section Universal synchronous asynchronous receiver transmitter (USART) of the corresponding reference manual.*

### 70.2.3 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 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible UART frame formats.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method

The HAL\_UART\_Init(), HAL\_HalfDuplex\_Init(), HAL\_LIN\_Init() and HAL\_MultiProcessor\_Init() APIs follow respectively the UART asynchronous, UART Half duplex, LIN and Multi-Processor configuration procedures (details for the procedures are available in reference manual (RM0430 for STM32F4X3xx MCUs and RM0402 for STM32F412xx MCUs RM0383 for STM32F411xC/E MCUs and RM0401 for STM32F410xx MCUs RM0090 for STM32F4X5xx/STM32F4X7xx/STM32F429xx/STM32F439xx MCUs RM0390 for STM32F446xx MCUs and RM0386 for STM32F469xx/STM32F479xx MCUs)).

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\(\)](#)

### 70.2.4 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\\_DMABasePause\(\)](#)
- [HAL\\_UART\\_DMABaseResume\(\)](#)
- [HAL\\_UART\\_DMABaseStop\(\)](#)
- [HAL\\_UARTEx\\_ReceiveTxdle\(\)](#)
- [HAL\\_UARTEx\\_ReceiveTxdle\\_IT\(\)](#)
- [HAL\\_UARTEx\\_ReceiveTxdle\\_DMA\(\)](#)
- [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()*
- *HAL\_UARTEEx\_RxEventCallback()*

### 70.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART:

- *HAL\_LIN\_SendBreak()* API can be helpful to transmit the break character.
- *HAL\_MultiProcessor\_EnterMuteMode()* API can be helpful to enter the UART in mute mode.
- *HAL\_MultiProcessor\_ExitMuteMode()* API can be helpful to exit the UART mute mode by software.
- *HAL\_HalfDuplex\_EnableTransmitter()* API to enable the UART transmitter and disables the UART receiver in Half Duplex mode
- *HAL\_HalfDuplex\_EnableReceiver()* API to enable the UART receiver and disables the UART transmitter in Half Duplex mode

This section contains the following APIs:

- *HAL\_LIN\_SendBreak()*
- *HAL\_MultiProcessor\_EnterMuteMode()*
- *HAL\_MultiProcessor\_ExitMuteMode()*
- *HAL\_HalfDuplex\_EnableTransmitter()*
- *HAL\_HalfDuplex\_EnableReceiver()*

### 70.2.6 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of UART communication process, return Peripheral Errors occurred during communication process

- *HAL\_UART\_GetState()* API can be helpful to check in run-time the state of the UART peripheral.
- *HAL\_UART\_GetError()* check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- *HAL\_UART\_GetState()*
- *HAL\_UART\_GetError()*

### 70.2.7 Detailed description of functions

#### HAL\_UART\_Init

##### Function name

**HAL\_StatusTypeDef HAL\_UART\_Init (UART\_HandleTypeDef \* huart)**

##### Function description

Initializes the UART mode according to the specified parameters in the UART\_InitTypeDef and create the associated handle.

##### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **HAL:** status

### HAL\_HalfDuplex\_Init

### Function name

**HAL\_StatusTypeDef HAL\_HalfDuplex\_Init (UART\_HandleTypeDef \* huart)**

### Function description

Initializes the half-duplex mode according to the specified parameters in the UART\_InitTypeDef and create the associated handle.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **HAL:** status

### HAL\_LIN\_Init

### Function name

**HAL\_StatusTypeDef HAL\_LIN\_Init (UART\_HandleTypeDef \* huart, uint32\_t BreakDetectLength)**

### Function description

Initializes the LIN mode according to the specified parameters in the UART\_InitTypeDef and create the associated handle.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **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

Initializes the Multi-Processor mode according to the specified parameters in the UART\_InitTypeDef and create the associated handle.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **Address:** USART address
- **WakeUpMethod:** specifies the USART wake-up method. This parameter can be one of the following values:
  - UART\_WAKEUPMETHOD\_IDLELINE: Wake-up by an idle line detection
  - UART\_WAKEUPMETHOD\_ADDRESSMARK: Wake-up by an address mark

**Return values**

- **HAL:** status

**HAL\_UART\_DeInit**
**Function name**
**HAL\_StatusTypeDef HAL\_UART\_DeInit (UART\_HandleTypeDef \* huart)**
**Function description**

DeInitializes the UART peripheral.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

**Return values**

- **HAL:** status

**HAL\_UART\_MspInit**
**Function name**
**void HAL\_UART\_MspInit (UART\_HandleTypeDef \* huart)**
**Function description**

UART MSP Init.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

**Return values**

- **None:**

**HAL\_UART\_MspDeInit**
**Function name**
**void HAL\_UART\_MspDeInit (UART\_HandleTypeDef \* huart)**
**Function description**

UART MSP DeInit.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

**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**

Sends an amount of data in blocking mode.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **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.

## HAL\_UART\_Receive

### Function name

**HAL\_StatusTypeDef HAL\_UART\_Receive (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

### Function description

Receives an amount of data in blocking mode.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **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.

## HAL\_UART\_Transmit\_IT

### Function name

**HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size)**

### Function description

Sends an amount of data in non blocking mode.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART 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 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**

Receives an amount of data in non blocking mode.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART 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 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**

Sends an amount of data in DMA mode.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART 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 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

Receives an amount of data in DMA mode.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART 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 indicate the number of u16 available through pData.
- When the UART parity is enabled (PCE = 1) the received data contains the parity bit.

### HAL\_UART\_DMAPause

#### Function name

**HAL\_StatusTypeDef HAL\_UART\_DMAPause (UART\_HandleTypeDef \* huart)**

#### Function description

Pauses the DMA Transfer.

#### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

#### Return values

- **HAL:** status

### HAL\_UART\_DMAResume

#### Function name

**HAL\_StatusTypeDef HAL\_UART\_DMAResume (UART\_HandleTypeDef \* huart)**

#### Function description

Resumes the DMA Transfer.

#### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

#### Return values

- **HAL:** status

### HAL\_UART\_DMAStop

#### Function name

**HAL\_StatusTypeDef HAL\_UART\_DMAStop (UART\_HandleTypeDef \* huart)**

#### Function description

Stops the DMA Transfer.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **HAL:** status

### HAL\_UARTEx\_ReceiveToldle

### Function name

**HAL\_StatusTypeDef HAL\_UARTEx\_ReceiveToldle (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size, uint16\_t \* RxLen, uint32\_t Timeout)**

### Function description

Receive an amount of data in blocking mode till either the expected number of data is received or an IDLE event occurs.

### Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8\_t or uint16\_t data elements).
- **Size:** Amount of data elements (uint8\_t or uint16\_t) to be received.
- **RxLen:** Number of data elements finally received (could be lower than Size, in case reception ends on IDLE event)
- **Timeout:** Timeout duration expressed in ms (covers the whole reception sequence).

### Return values

- **HAL:** status

### Notes

- HAL\_OK is returned if reception is completed (expected number of data has been received) or if reception is stopped after IDLE event (less than the expected number of data has been received) In this case, RxLen output parameter indicates number of data available in reception buffer.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M = 01), the received data is handled as a set of uint16\_t. In this case, Size must indicate the number of uint16\_t available through pData.

### HAL\_UARTEx\_ReceiveToldle\_IT

### Function name

**HAL\_StatusTypeDef HAL\_UARTEx\_ReceiveToldle\_IT (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size)**

### Function description

Receive an amount of data in interrupt mode till either the expected number of data is received or an IDLE event occurs.

### Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8\_t or uint16\_t data elements).
- **Size:** Amount of data elements (uint8\_t or uint16\_t) to be received.

### Return values

- **HAL:** status



## Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to UART interrupts raised by RXNE and IDLE events. Callback is called at end of reception indicating number of received data elements.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M = 01), the received data is handled as a set of uint16\_t. In this case, Size must indicate the number of uint16\_t available through pData.

### HAL\_UARTEx\_ReceiveToldle\_DMA

#### Function name

**HAL\_StatusTypeDef HAL\_UARTEx\_ReceiveToldle\_DMA (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size)**

#### Function description

Receive an amount of data in DMA mode till either the expected number of data is received or an IDLE event occurs.

#### Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8\_t or uint16\_t data elements).
- **Size:** Amount of data elements (uint8\_t or uint16\_t) to be received.

#### Return values

- **HAL:** status

## Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to DMA services, transferring automatically received data elements in user reception buffer and calling registered callbacks at half/end of reception. UART IDLE events are also used to consider reception phase as ended. In all cases, callback execution will indicate number of received data elements.
- 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 (M = 01), the received data is handled as a set of uint16\_t. In this case, Size must indicate the number of uint16\_t available through pData.

### 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

This function handles UART interrupt request.

### Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **None**:

**HAL\_UART\_TxCpltCallback**

### Function name

**void HAL\_UART\_TxCpltCallback (UART\_HandleTypeDef \* huart)**

### Function description

Tx Transfer completed callbacks.

### Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **None**:

**HAL\_UART\_TxHalfCpltCallback**

### Function name

**void HAL\_UART\_TxHalfCpltCallback (UART\_HandleTypeDef \* huart)**

### Function description

Tx Half Transfer completed callbacks.

### Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **None**:

**HAL\_UART\_RxCpltCallback**

### Function name

**void HAL\_UART\_RxCpltCallback (UART\_HandleTypeDef \* huart)**

### Function description

Rx Transfer completed callbacks.

### Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **None**:

**HAL\_UART\_RxHalfCpltCallback**

### Function name

**void HAL\_UART\_RxHalfCpltCallback (UART\_HandleTypeDef \* huart)**

### Function description

Rx Half Transfer completed callbacks.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **None:**

**HAL\_UART\_ErrorCallback**

### Function name

**void HAL\_UART\_ErrorCallback (UART\_HandleTypeDef \* huart)**

### Function description

UART error callbacks.

### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### 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\_UARTEEx\_RxEventCallback**
**Function name**

**void HAL\_UARTEEx\_RxEventCallback (UART\_HandleTypeDef \* huart, uint16\_t Size)**

**Function description**

Reception Event Callback (Rx event notification called after use of advanced reception service).

**Parameters**

- **huart:** UART handle
- **Size:** Number of data available in application reception buffer (indicates a position in reception buffer until which, data are available)

**Return values**

- **None:**

**HAL\_LIN\_SendBreak**
**Function name**

**HAL\_StatusTypeDef HAL\_LIN\_SendBreak (UART\_HandleTypeDef \* huart)**

**Function description**

Transmits break characters.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

**Return values**

- **HAL:** status

**HAL\_MultiProcessor\_EnterMuteMode**
**Function name**

**HAL\_StatusTypeDef HAL\_MultiProcessor\_EnterMuteMode (UART\_HandleTypeDef \* huart)**

**Function description**

Enters the UART in mute mode.

**Parameters**

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

**Return values**

- **HAL:** status

### HAL\_MultiProcessor\_ExitMuteMode

#### Function name

**HAL\_StatusTypeDef HAL\_MultiProcessor\_ExitMuteMode (UART\_HandleTypeDef \* huart)**

#### Function description

Exits the UART mute mode: wake up software.

#### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

#### Return values

- **HAL:** status

### HAL\_HalfDuplex\_EnableTransmitter

#### Function name

**HAL\_StatusTypeDef HAL\_HalfDuplex\_EnableTransmitter (UART\_HandleTypeDef \* huart)**

#### Function description

Enables the UART transmitter and disables the UART receiver.

#### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

#### Return values

- **HAL:** status

### HAL\_HalfDuplex\_EnableReceiver

#### Function name

**HAL\_StatusTypeDef HAL\_HalfDuplex\_EnableReceiver (UART\_HandleTypeDef \* huart)**

#### Function description

Enables the UART receiver and disables the UART transmitter.

#### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

#### Return values

- **HAL:** status

### HAL\_UART\_GetState

#### Function name

**HAL\_UART\_StateTypeDef HAL\_UART\_GetState (UART\_HandleTypeDef \* huart)**

#### Function description

Returns the UART state.

#### Parameters

- **huart:** Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

### Return values

- **HAL:** state

### HAL\_UART\_GetError

### Function name

**uint32\_t HAL\_UART\_GetError (UART\_HandleTypeDef \* huart)**

### Function description

Return the UART 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\_Start\_Receive\_IT

### Function name

**HAL\_StatusTypeDef UART\_Start\_Receive\_IT (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size)**

### Function description

Start Receive operation 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

- This function could be called by all HAL UART API providing reception in Interrupt mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

### UART\_Start\_Receive\_DMA

### Function name

**HAL\_StatusTypeDef UART\_Start\_Receive\_DMA (UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size)**

### Function description

Start Receive operation 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

- This function could be called by all HAL UART API providing reception in DMA mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

## 70.3 UART Firmware driver defines

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

### 70.3.1 UART

UART

#### *UART Error Code*

#### 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

#### *UART Exported Macros*

#### \_\_HAL\_UART\_RESET\_HANDLE\_STATE

##### **Description:**

- Reset UART handle gstate & RxState.

##### **Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

##### **Return value:**

- None

#### \_\_HAL\_UART\_FLUSH\_DRREGISTER

##### **Description:**

- Flushes the UART DR register.

##### **Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

### **\_\_HAL\_UART\_GET\_FLAG**

**Description:**

- Checks whether the specified UART flag is set or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - **UART\_FLAG\_CTS**: CTS Change flag (not available for UART4 and UART5)
  - **UART\_FLAG\_LBD**: LIN Break detection flag
  - **UART\_FLAG\_TXE**: Transmit data register empty flag
  - **UART\_FLAG\_TC**: Transmission Complete flag
  - **UART\_FLAG\_RXNE**: Receive data register not empty 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\_CLEAR\_FLAG**

**Description:**

- Clears the specified UART pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be any combination of the following values:
  - **UART\_FLAG\_CTS**: CTS Change flag (not available for UART4 and UART5).
  - **UART\_FLAG\_LBD**: LIN Break detection flag.
  - **UART\_FLAG\_TC**: Transmission Complete flag.
  - **UART\_FLAG\_RXNE**: Receive data register not empty flag.

**Return value:**

- None

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (Overrun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

### **\_\_HAL\_UART\_CLEAR\_PFLAG**

**Description:**

- Clears the UART PE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### **\_\_HAL\_UART\_CLEAR\_FEFLAG**

**Description:**

- Clears the UART FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### **\_\_HAL\_UART\_CLEAR\_NEFLAG**

**Description:**

- Clears the UART NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### **\_\_HAL\_UART\_CLEAR\_OREFLAG**

**Description:**

- Clears the UART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### **\_\_HAL\_UART\_CLEAR\_IDLEFLAG**

**Description:**

- Clears the UART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

### \_\_HAL\_UART\_ENABLE\_IT

**Description:**

- Enable the specified UART interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- \_\_INTERRUPT\_\_: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty 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. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- \_\_INTERRUPT\_\_: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty 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\_SOURCE**

**Description:**

- Checks whether the specified UART interrupt source is enabled or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_IT\_\_**: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - **UART\_IT\_CTS**: CTS change interrupt (not available for UART4 and UART5)
  - **UART\_IT\_LBD**: LIN Break detection interrupt
  - **UART\_IT\_TXE**: Transmit Data Register empty interrupt
  - **UART\_IT\_TC**: Transmission complete interrupt
  - **UART\_IT\_RXNE**: Receive Data register not empty interrupt
  - **UART\_IT\_IDLE**: Idle line detection interrupt
  - **UART\_IT\_ERR**: Error interrupt

**Return value:**

- The: new state of **\_\_IT\_\_** (TRUE or FALSE).

### **\_\_HAL\_UART\_HWCONTROL\_CTS\_ENABLE**

**Description:**

- Enable CTS flow control.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

**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. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

**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 This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

**Return value:**

- None

**Notes:**

- 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 This macro allows to disable RTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. The Handle Instance can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

**Return value:**

- None

**Notes:**

- 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\_ONE\_BIT\_SAMPLE\_ENABLE

**Description:**

- Macro to enable the UART's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

#### \_\_HAL\_UART\_ONE\_BIT\_SAMPLE\_DISABLE

**Description:**

- Macro to disable the UART's 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

**UART FLags**

UART\_FLAG\_CTS

UART\_FLAG\_LBD

UART\_FLAG\_TXE

UART\_FLAG\_TC

UART\_FLAG\_RXNE

UART\_FLAG\_IDLE

UART\_FLAG\_ORE

UART\_FLAG\_NE

UART\_FLAG\_FE

UART\_FLAG\_PE

*UART Hardware Flow Control*

UART\_HWCONTROL\_NONE

UART\_HWCONTROL\_RTS

UART\_HWCONTROL\_CTS

UART\_HWCONTROL\_RTS\_CTS

*UART Interrupt Definitions*

UART\_IT\_PE

UART\_IT\_TXE

UART\_IT\_TC

UART\_IT\_RXNE

UART\_IT\_IDLE

UART\_IT\_LBD

UART\_IT\_CTS

UART\_IT\_ERR

*UART LIN Break Detection Length*

UART\_LINBREAKDETECTLENGTH\_10B

UART\_LINBREAKDETECTLENGTH\_11B

*UART Transfer Mode*

UART\_MODE\_RX

UART\_MODE\_TX

UART\_MODE\_TX\_RX

*UART Over Sampling*

UART\_OVERSAMPLING\_16

UART\_OVERSAMPLING\_8

*UART Parity*

UART\_PARITY\_NONE

UART\_PARITY\_EVEN

UART\_PARITY\_ODD

*UART Reception type values*

HAL\_UART\_RECEPTION\_STANDARD

Standard reception



HAL\_UART\_RECEPTION\_TOIDLE

Reception till completion or IDLE event

**UART State**

UART\_STATE\_DISABLE

UART\_STATE\_ENABLE

**UART Number of Stop Bits**

UART\_STOPBITS\_1

UART\_STOPBITS\_2

**UART Wakeup Functions**

UART\_WAKEUPMETHOD\_IDLELINE

UART\_WAKEUPMETHOD\_ADDRESSMARK

**UART Word Length**

UART\_WORDLENGTH\_8B

UART\_WORDLENGTH\_9B

## 71 HAL USART Generic Driver

### 71.1 USART Firmware driver registers structures

#### 71.1.1 USART\_InitTypeDef

*USART\_InitTypeDef* is defined in the `stm32f4xx_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*

##### Field Documentation

- *uint32\_t* *USART\_InitTypeDef::BaudRate*  
This member configures the Usart communication baud rate. The baud rate is computed using the following formula:
  - $IntegerDivider = ((PCLKx) / (8 * (husart->Init.BaudRate)))$
  - $FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 8) + 0.5$
- *uint32\_t* *USART\_InitTypeDef::WordLength*  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [USART\\_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](#)

#### 71.1.2 \_\_USART\_HandleTypeDef

*\_\_USART\_HandleTypeDef* is defined in the `stm32f4xx_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`
- `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
- **`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

## 71.2 USART Firmware driver API description

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

### 71.2.1 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:
  - a. Enable the USARTx interface clock.
  - b. USART pins configuration:
    - Enable the clock for the USART GPIOs.
    - Configure the USART pins as alternate function pull-up.
  - c. 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.
  - d. 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 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 USART DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx stream.
    - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the husart Init structure.
4. Initialize the USART registers by calling the HAL\_USART\_Init() API:
  - These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_USART\_MspInit(&husart) API.

*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.

5. Three operation modes are available within this driver :

#### **Polling mode IO operation**

- Send an amount of data in blocking mode using HAL\_USART\_Transmit()
- Receive an amount of data in blocking mode using HAL\_USART\_Receive()

#### **Interrupt mode IO operation**

- Send an amount of data in non blocking mode using HAL\_USART\_Transmit\_IT()
- At transmission end of transfer HAL\_USART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_USART\_Receive\_IT()
- At reception end of transfer HAL\_USART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxCpltCallback
- In case of transfer Error, HAL\_USART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_USART\_ErrorCallback

#### **DMA mode IO operation**

- Send an amount of data in non blocking mode (DMA) using HAL\_USART\_Transmit\_DMA()
- At transmission end of half transfer HAL\_USART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_USART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_USART\_Receive\_DMA()
- At reception end of half transfer HAL\_USART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxHalfCpltCallback

- At reception end of transfer HAL\_USART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxCpltCallback
- In case of transfer Error, HAL\_USART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_USART\_ErrorCallback
- Pause the DMA Transfer using HAL\_USART\_DMAMPause()
- Resume the DMA Transfer using HAL\_USART\_DMAResume()
- Stop the DMA Transfer using HAL\_USART\_DMAStop()

### USART HAL driver macros list

Below the list of most used macros in USART HAL driver.

- `__HAL_USART_ENABLE`: Enable the USART peripheral
- `__HAL_USART_DISABLE`: Disable the USART peripheral
- `__HAL_USART_GET_FLAG` : Check whether the specified USART flag is set or not
- `__HAL_USART_CLEAR_FLAG` : Clear the specified USART pending flag
- `__HAL_USART_ENABLE_IT`: Enable the specified USART interrupt
- `__HAL_USART_DISABLE_IT`: Disable the specified USART interrupt

*Note:* You can refer to the USART HAL driver header file for more useful macros

## 71.2.2 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.
- `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.
- `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.

*Note: If the parity is enabled, the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. The USART frame format depends on the frame length defined by the M bit (8-bits or 9-bits). For more details, refer to Table Frame formats in Section Universal synchronous asynchronous receiver transmitter (USART) of the corresponding reference manual.*

### 71.2.3 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible USART frame formats.
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL\_USART\_Init() function follows the USART synchronous configuration procedures (details for the procedures are available in reference manual (RM0430 for STM32F4X3xx MCUs and RM0402 for STM32F412xx MCUs RM0383 for STM32F411xC/E MCUs and RM0401 for STM32F410xx MCUs RM0090 for STM32F4X5xx/ STM32F4X7xx/STM32F429xx/STM32F439xx MCUs RM0390 for STM32F446xx MCUs and RM0386 for STM32F469xx/STM32F479xx MCUs)).

This section contains the following APIs:

- [HAL\\_USART\\_Init\(\)](#)
- [HAL\\_USART\\_DeInit\(\)](#)
- [HAL\\_USART\\_MspInit\(\)](#)
- [HAL\\_USART\\_MspDeInit\(\)](#)

### 71.2.4 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 APIs 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 APIs 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. Non Blocking mode functions 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\_TxHalfCpltCallback()
  - HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxHalfCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - 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\(\)](#)

### 71.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of USART communication process, return Peripheral Errors occurred during communication process

- [HAL\\_USART\\_GetState\(\)](#) API can be helpful to check in run-time the state of the USART peripheral.
- [HAL\\_USART\\_GetError\(\)](#) check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL\\_USART\\_GetState\(\)](#)
- [HAL\\_USART\\_GetError\(\)](#)

### 71.2.6 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**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

##### Return values

- **HAL**: status

#### HAL\_USART\_DeInit

##### Function name

**HAL\_StatusTypeDef HAL\_USART\_DeInit (USART\_HandleTypeDef \* husart)**

##### Function description

Deinitializes the USART peripheral.

##### Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

##### Return values

- **HAL**: status

#### HAL\_USART\_MspInit

##### Function name

**void HAL\_USART\_MspInit (USART\_HandleTypeDef \* husart)**

##### Function description

USART MSP Init.



### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

### Return values

- **None:**

### HAL\_USART\_MspDeInit

### Function name

**void HAL\_USART\_MspDeInit (USART\_HandleTypeDef \* husart)**

### Function description

USART MSP DeInit.

### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

### 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:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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

Full-Duplex Receive an amount of data in blocking mode.

### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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 full-duplex mode (blocking mode).

### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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

Simplex Send an amount of data in non-blocking mode.

### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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.
- The USART errors are not managed to avoid the overrun error.

### HAL\_USART\_Receive\_IT

#### Function name

**HAL\_StatusTypeDef HAL\_USART\_Receive\_IT (USART\_HandleTypeDef \* husart, uint8\_t \* pRxData, uint16\_t Size)**

#### Function description

Simplex Receive an amount of data in non-blocking mode.

#### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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 full-duplex mode (non-blocking).

#### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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

Simplex Send an amount of data in DMA mode.

#### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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

Full-Duplex Receive an amount of data in DMA mode.

#### Parameters

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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 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.
- The USART DMA transmit stream must be configured in order to generate the clock for the slave.
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

### 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 DMA mode.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **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 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.
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

**HAL\_USART\_DMAPause**
**Function name**

**HAL\_StatusTypeDef HAL\_USART\_DMAPause (USART\_HandleTypeDef \* husart)**

**Function description**

Pauses the DMA Transfer.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **HAL:** status

**HAL\_USART\_DMAResume**
**Function name**

**HAL\_StatusTypeDef HAL\_USART\_DMAResume (USART\_HandleTypeDef \* husart)**

**Function description**

Resumes the DMA Transfer.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **HAL:** status

**HAL\_USART\_DMAStop**
**Function name**

**HAL\_StatusTypeDef HAL\_USART\_DMAStop (USART\_HandleTypeDef \* husart)**

**Function description**

Stops the DMA Transfer.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **HAL:** status

**HAL\_USART\_Abort**
**Function name**
**HAL\_StatusTypeDef HAL\_USART\_Abort (USART\_HandleTypeDef \* husart)**
**Function description**

Abort ongoing transfer (blocking mode).

**Parameters**

- **husart:** USART handle.

**Return values**

- **HAL:** status

**Notes**

- This procedure could be used for aborting any ongoing transfer (either Tx or Rx, as described by TransferType parameter) started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP 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\_USART\_Abort\_IT**
**Function name**
**HAL\_StatusTypeDef HAL\_USART\_Abort\_IT (USART\_HandleTypeDef \* husart)**
**Function description**

Abort ongoing transfer (Interrupt mode).

**Parameters**

- **husart:** USART handle.

**Return values**

- **HAL:** status

**Notes**

- This procedure could be used for aborting any ongoing transfer (either Tx or Rx, as described by TransferType parameter) started in Interrupt or DMA mode. This procedure performs following operations : Disable PPP 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\_USART\_IRQHandler**
**Function name**
**void HAL\_USART\_IRQHandler (USART\_HandleTypeDef \* husart)**
**Function description**

This function handles USART interrupt request.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **None:**

**HAL\_USART\_TxCpltCallback**

**Function name**

**void HAL\_USART\_TxCpltCallback (USART\_HandleTypeDef \* husart)**

**Function description**

Tx Transfer completed callbacks.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **None:**

**HAL\_USART\_TxHalfCpltCallback**

**Function name**

**void HAL\_USART\_TxHalfCpltCallback (USART\_HandleTypeDef \* husart)**

**Function description**

Tx Half Transfer completed callbacks.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **None:**

**HAL\_USART\_RxCpltCallback**

**Function name**

**void HAL\_USART\_RxCpltCallback (USART\_HandleTypeDef \* husart)**

**Function description**

Rx Transfer completed callbacks.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **None:**

**HAL\_USART\_RxHalfCpltCallback**

**Function name**

**void HAL\_USART\_RxHalfCpltCallback (USART\_HandleTypeDef \* husart)**

**Function description**

Rx Half Transfer completed callbacks.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**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:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**Return values**

- **None:**

**HAL\_USART\_ErrorCallback**
**Function name**

```
void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)
```

**Function description**

USART error callbacks.

**Parameters**

- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

**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**

Returns the USART state.



#### Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

#### Return values

- **HAL**: 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**: Error Code

## 71.3 USART Firmware driver defines

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

### 71.3.1 USART

USART  
*USART Clock*

**USART\_CLOCK\_DISABLE**

**USART\_CLOCK\_ENABLE**

*USART Clock Phase*

**USART\_PHASE\_1EDGE**

**USART\_PHASE\_2EDGE**

*USART Clock Polarity*

**USART\_POLARITY\_LOW**

**USART\_POLARITY\_HIGH**

*USART Error Code*

**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

### *USART Exported Macros*

### \_\_HAL\_USART\_RESET\_HANDLE\_STATE

**Description:**

- Reset USART handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### \_\_HAL\_USART\_GET\_FLAG

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - USART\_FLAG\_TXE: Transmit data register empty flag
  - USART\_FLAG\_TC: Transmission Complete flag
  - USART\_FLAG\_RXNE: Receive data register not empty 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 flags.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be any combination of the following values:
  - USART\_FLAG\_TC: Transmission Complete flag.
  - USART\_FLAG\_RXNE: Receive data register not empty flag.

**Return value:**

- None

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (Overrun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

### **\_\_HAL\_USART\_CLEAR\_PFLAG**

**Description:**

- Clear the USART PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_USART\_CLEAR\_FEFLAG**

**Description:**

- Clear the USART FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_USART\_CLEAR\_NEFLAG**

**Description:**

- Clear the USART NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_USART\_CLEAR\_OREFLAG**

**Description:**

- Clear the USART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_USART\_CLEAR\_IDLEFLAG**

**Description:**

- Clear the USART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### \_\_HAL\_USART\_ENABLE\_IT

**Description:**

- Enables or disables the specified USART interrupts.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__INTERRUPT__`: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register 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

### \_\_HAL\_USART\_GET\_IT\_SOURCE

**Description:**

- Checks whether the specified USART interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__IT__`: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ERR: Error interrupt
  - USART\_IT\_PE: Parity Error interrupt

**Return value:**

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

### \_\_HAL\_USART\_ONE\_BIT\_SAMPLE\_ENABLE

**Description:**

- Macro to enable the USART's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

### \_\_HAL\_USART\_ONE\_BIT\_SAMPLE\_DISABLE

**Description:**

- Macro to disable the USART's 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. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

### **\_\_HAL\_USART\_DISABLE**

**Description:**

- Disable USART.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

**USART Flags**

**USART\_FLAG\_TXE**

**USART\_FLAG\_TC**

**USART\_FLAG\_RXNE**

**USART\_FLAG\_IDLE**

**USART\_FLAG\_ORE**

**USART\_FLAG\_NE**

**USART\_FLAG\_FE**

**USART\_FLAG\_PE**

**USART Interrupts Definition**

**USART\_IT\_PE**

**USART\_IT\_TXE**

**USART\_IT\_TC**

**USART\_IT\_RXNE**

**USART\_IT\_IDLE**

**USART\_IT\_ERR**

**USART Last Bit**

**USART\_LASTBIT\_DISABLE**

**USART\_LASTBIT\_ENABLE**

**USART Mode**

USART\_MODE\_RX

USART\_MODE\_TX

USART\_MODE\_TX\_RX

*USART NACK State*

USART\_NACK\_ENABLE

USART\_NACK\_DISABLE

*USART Parity*

USART\_PARITY\_NONE

USART\_PARITY\_EVEN

USART\_PARITY\_ODD

*USART Number of Stop Bits*

USART\_STOPBITS\_1

USART\_STOPBITS\_0\_5

USART\_STOPBITS\_2

USART\_STOPBITS\_1\_5

*USART Word Length*

USART\_WORDLENGTH\_8B

USART\_WORDLENGTH\_9B

## 72 HAL WWDG Generic Driver

### 72.1 WWDG Firmware driver registers structures

#### 72.1.1 WWDG\_InitTypeDef

*WWDG\_InitTypeDef* is defined in the `stm32f4xx_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](#)

#### 72.1.2 WWDG\_HandleTypeDef

*WWDG\_HandleTypeDef* is defined in the `stm32f4xx_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

### 72.2 WWDG Firmware driver API description

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

#### 72.2.1 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\(\)](#)

#### 72.2.2 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\(\)](#)

### 72.2.3 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:**

## 72.3 WWDG Firmware driver defines

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

### 72.3.1

#### **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 =  $(PCLK1/4096)/8$

## 73 LL ADC Generic Driver

### 73.1 ADC Firmware driver registers structures

#### 73.1.1 LL\_ADC\_CommonInitTypeDef

*LL\_ADC\_CommonInitTypeDef* is defined in the `stm32f4xx_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](#)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()`.

#### 73.1.2 LL\_ADC\_InitTypeDef

*LL\_ADC\_InitTypeDef* is defined in the `stm32f4xx_ll_adc.h`

##### Data Fields

- *uint32\_t* `Resolution`
- *uint32\_t* `DataAlignment`
- *uint32\_t* `SequencersScanMode`

##### 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::SequencersScanMode`  
Set ADC scan selection. This parameter can be a value of [ADC\\_LL\\_EC\\_SCAN\\_SELECTION](#)This feature can be modified afterwards using unitary function `LL_ADC_SetSequencersScanMode()`.

#### 73.1.3 LL\_ADC\_REG\_InitTypeDef

*LL\_ADC\_REG\_InitTypeDef* is defined in the `stm32f4xx_ll_adc.h`

##### Data Fields

- *uint32\_t* `TriggerSource`
- *uint32\_t* `SequencerLength`

- *uint32\_t SequencerDiscont*
- *uint32\_t ContinuousMode*
- *uint32\_t DMATransfer*

#### Field Documentation

- *uint32\_t LL\_ADC\_REG\_InitTypeDef::TriggerSource*  
Set ADC group regular conversion trigger source: internal (SW start) or from external IP (timer event, external interrupt line). This parameter can be a value of [ADC\\_LL\\_EC\\_REG\\_TRIGGER\\_SOURCE](#)  
**Note:**
  - On this STM32 serie, setting of external trigger edge is performed using function `LL_ADC_REG_StartConversionExtTrig()`.

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](#)  
**Note:**
  - This parameter is discarded if scan mode is disabled (refer to parameter 'ADC\_SequencersScanMode').

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()`.

### 73.1.4

#### LL\_ADC\_INJ\_InitTypeDef

`LL_ADC_INJ_InitTypeDef` is defined in the `stm32f4xx_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 IP (timer event, external interrupt line). This parameter can be a value of [ADC\\_LL\\_EC\\_INJ\\_TRIGGER\\_SOURCE](#)  
**Note:**
  - On this STM32 serie, setting of external trigger edge is performed using function [LL\\_ADC\\_INJ\\_StartConversionExtTrig\(\)](#).
 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](#)  
**Note:**
  - This parameter is discarded if scan mode is disabled (refer to parameter 'ADC\_SequencersScanMode').
 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 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\(\)](#).

## 73.2 ADC Firmware driver API description

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

### 73.2.1 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\_DIV2
  - LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV4
  - LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV6
  - LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV8

#### Return values

- **None**:

## Reference Manual to LL API cross reference:

- CCR ADCPRE 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\_DIV2
  - LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV4
  - LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV6
  - LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV8

### Reference Manual to LL API cross reference:

- CCR ADCPRE 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)`
- 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 TSVREFE LL\_ADC\_SetCommonPathInternalCh
- CCR VBATE LL\_ADC\_SetCommonPathInternalCh

### 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 TSVREFE `LL_ADC_GetCommonPathInternalCh`
- CCR VBATE `LL_ADC_GetCommonPathInternalCh`

### **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:**

### Reference Manual to LL API cross reference:

- CR1 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:

- CR1 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.

### Reference Manual to LL API cross reference:

- CR2 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:

- CR2 ALIGN LL\_ADC\_SetDataAlignment

## LL\_ADC\_SetSequencersScanMode

### Function name

```
__STATIC_INLINE void LL_ADC_SetSequencersScanMode (ADC_TypeDef * ADCx, uint32_t ScanMode)
```

### Function description

Set ADC sequencers scan mode, for all ADC groups (group regular, group injected).

### Parameters

- **ADCx:** ADC instance
- **ScanMode:** This parameter can be one of the following values:
  - LL\_ADC\_SEQ\_SCAN\_DISABLE
  - LL\_ADC\_SEQ\_SCAN\_ENABLE

### Return values

- **None:**

### Notes

- According to sequencers scan mode : If disabled: ADC conversion is performed in unitary conversion mode (one channel converted, that defined in rank 1). Configuration of sequencers of all ADC groups (sequencer scan length, ...) is discarded: equivalent to scan length of 1 rank. If enabled: ADC conversions are performed in sequence conversions mode, according to configuration of sequencers of each ADC group (sequencer scan length, ...). Refer to function LL\_ADC\_REG\_SetSequencerLength() and to function LL\_ADC\_INJ\_SetSequencerLength().

### Reference Manual to LL API cross reference:

- CR1 SCAN LL\_ADC\_SetSequencersScanMode

## LL\_ADC\_GetSequencersScanMode

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_GetSequencersScanMode (ADC_TypeDef * ADCx)
```

### Function description

Get ADC sequencers scan mode, for all ADC groups (group regular, group injected).

### Parameters

- **ADCx:** ADC instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_ADC\_SEQ\_SCAN\_DISABLE
  - LL\_ADC\_SEQ\_SCAN\_ENABLE

### Notes

- According to sequencers scan mode : If disabled: ADC conversion is performed in unitary conversion mode (one channel converted, that defined in rank 1). Configuration of sequencers of all ADC groups (sequencer scan length, ...) is discarded: equivalent to scan length of 1 rank. If enabled: ADC conversions are performed in sequence conversions mode, according to configuration of sequencers of each ADC group (sequencer scan length, ...). Refer to function LL\_ADC\_REG\_SetSequencerLength() and to function LL\_ADC\_INJ\_SetSequencerLength().

### Reference Manual to LL API cross reference:

- CR1 SCAN LL\_ADC\_GetSequencersScanMode

## 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 IP (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\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM1\_CH2
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM1\_CH3
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH2
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH3
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH4
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_TRGO
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM3\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM3\_TRGO
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM4\_CH4
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH2
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH3
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM8\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM8\_TRGO
  - LL\_ADC\_REG\_TRIG\_EXT\_EXTI\_LINE11

### Return values

- **None:**

### Notes

- On this STM32 serie, setting of external trigger edge is performed using function LL\_ADC\_REG\_StartConversionExtTrig().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

### Reference Manual to LL API cross reference:

- CR2 EXTSEL LL\_ADC\_REG\_SetTriggerSource
- CR2 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 IP (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\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM1\_CH2
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM1\_CH3
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH2
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH3
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH4
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_TRGO
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM3\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM3\_TRGO
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM4\_CH4
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH2
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH3
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM8\_CH1
  - LL\_ADC\_REG\_TRIG\_EXT\_TIM8\_TRGO
  - LL\_ADC\_REG\_TRIG\_EXT\_EXTI\_LINE11

### 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:

- CR2 EXTSEL LL\_ADC\_REG\_GetTriggerSource
- CR2 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:

- CR2 EXTEN LL\_ADC\_REG\_IsTriggerSourceSWStart

## 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.
- On this STM32 serie, setting of external trigger edge is performed using function LL\_ADC\_REG\_StartConversionExtTrig().

### Reference Manual to LL API cross reference:

- CR2 EXTEN LL\_ADC\_REG\_GetTriggerEdge

## 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()".
- On this STM32 serie, group regular sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL\_ADC\_SetSequencersScanMode().
- 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\_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()".
- On this STM32 serie, group regular sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL\_ADC\_SetSequencersScanMode().
- 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\_SetSequencerLength

## 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.

## Reference Manual to LL API cross reference:

- CR1 DISCEN LL\_ADC\_REG\_SetSequencerDiscont
- CR1 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:

- CR1 DISCEN LL\_ADC\_REG\_GetSequencerDiscont
- CR1 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
  - 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
  - LL\_ADC\_CHANNEL\_VREFINT (1)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
  - LL\_ADC\_CHANNEL\_VBAT (1)

(1) On STM32F4, parameter available only on ADC instance: ADC1.

(2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

## 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()`.

**Reference Manual to LL API cross reference:**

- SQR3 SQ1 `LL_ADC_REG_SetSequencerRanks`
- SQR3 SQ2 `LL_ADC_REG_SetSequencerRanks`
- SQR3 SQ3 `LL_ADC_REG_SetSequencerRanks`
- SQR3 SQ4 `LL_ADC_REG_SetSequencerRanks`
- SQR3 SQ5 `LL_ADC_REG_SetSequencerRanks`
- SQR3 SQ6 `LL_ADC_REG_SetSequencerRanks`
- SQR2 SQ7 `LL_ADC_REG_SetSequencerRanks`
- SQR2 SQ8 `LL_ADC_REG_SetSequencerRanks`
- SQR2 SQ9 `LL_ADC_REG_SetSequencerRanks`
- SQR2 SQ10 `LL_ADC_REG_SetSequencerRanks`
- SQR2 SQ11 `LL_ADC_REG_SetSequencerRanks`
- SQR2 SQ12 `LL_ADC_REG_SetSequencerRanks`
- SQR1 SQ13 `LL_ADC_REG_SetSequencerRanks`
- SQR1 SQ14 `LL_ADC_REG_SetSequencerRanks`
- SQR1 SQ15 `LL_ADC_REG_SetSequencerRanks`
- SQR1 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
    - 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
    - LL\_ADC\_CHANNEL\_VREFINT (1)
    - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
    - LL\_ADC\_CHANNEL\_VBAT (1)
- (1) On STM32F4, parameter available only on ADC instance: ADC1.
- (2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.
  - (1) 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:**

- SQR3 SQ1 `LL_ADC_REG_GetSequencerRanks`
- SQR3 SQ2 `LL_ADC_REG_GetSequencerRanks`
- SQR3 SQ3 `LL_ADC_REG_GetSequencerRanks`
- SQR3 SQ4 `LL_ADC_REG_GetSequencerRanks`
- SQR3 SQ5 `LL_ADC_REG_GetSequencerRanks`
- SQR3 SQ6 `LL_ADC_REG_GetSequencerRanks`
- SQR2 SQ7 `LL_ADC_REG_GetSequencerRanks`
- SQR2 SQ8 `LL_ADC_REG_GetSequencerRanks`
- SQR2 SQ9 `LL_ADC_REG_GetSequencerRanks`
- SQR2 SQ10 `LL_ADC_REG_GetSequencerRanks`
- SQR2 SQ11 `LL_ADC_REG_GetSequencerRanks`
- SQR2 SQ12 `LL_ADC_REG_GetSequencerRanks`
- SQR1 SQ13 `LL_ADC_REG_GetSequencerRanks`
- SQR1 SQ14 `LL_ADC_REG_GetSequencerRanks`
- SQR1 SQ15 `LL_ADC_REG_GetSequencerRanks`
- SQR1 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 trigger; continuous 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.

**Reference Manual to LL API cross reference:**

- CR2 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:

- CR2 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().

**Reference Manual to LL API cross reference:**

- CR2 DMA LL\_ADC\_REG\_SetDMATransfer
- CR2 DDS 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:**

- CR2 DMA LL\_ADC\_REG\_GetDMATransfer
- CR2 DDS LL\_ADC\_REG\_GetDMATransfer

**LL\_ADC\_REG\_SetFlagEndOfConversion**
**Function name**

```
__STATIC_INLINE void LL_ADC_REG_SetFlagEndOfConversion (ADC_TypeDef * ADCx, uint32_t EocSelection)
```

**Function description**

Specify which ADC flag between EOC (end of unitary conversion) or EOS (end of sequence conversions) is used to indicate the end of conversion.

**Parameters**

- **ADCx:** ADC instance
- **EocSelection:** This parameter can be one of the following values:
  - LL\_ADC\_REG\_FLAG\_EOC\_SEQUENCE\_CONV
  - LL\_ADC\_REG\_FLAG\_EOC\_UNITARY\_CONV

**Return values**

- **None:**

### Notes

- This feature is aimed to be set when using ADC with programming model by polling or interruption (programming model by DMA usually uses DMA interruptions to indicate end of conversion and data transfer).
- For ADC group injected, end of conversion (flag&IT) is raised only at the end of the sequence.

### Reference Manual to LL API cross reference:

- CR2 EOCS LL\_ADC\_REG\_SetFlagEndOfConversion

#### LL\_ADC\_REG\_GetFlagEndOfConversion

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_ADC\_REG\_GetFlagEndOfConversion (ADC\_TypeDef \* ADCx)**

### Function description

Get which ADC flag between EOC (end of unitary conversion) or EOS (end of sequence conversions) is used to indicate the end of conversion.

### Parameters

- **ADCx:** ADC instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_ADC\_REG\_FLAG\_EOC\_SEQUENCE\_CONV
  - LL\_ADC\_REG\_FLAG\_EOC\_UNITARY\_CONV

### Reference Manual to LL API cross reference:

- CR2 EOCS LL\_ADC\_REG\_GetFlagEndOfConversion

#### 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 IP (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\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM1\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM2\_CH1
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM2\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM3\_CH2
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM3\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH1
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH2
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH3
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM5\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM5\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH2
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH3
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_EXTI\_LINE15

### Return values

- **None:**

### Notes

- On this STM32 serie, setting of external trigger edge is performed using function LL\_ADC\_INJ\_StartConversionExtTrig().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

### Reference Manual to LL API cross reference:

- CR2 JEXTSEL LL\_ADC\_INJ\_SetTriggerSource
- CR2 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 IP (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\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM1\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM2\_CH1
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM2\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM3\_CH2
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM3\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH1
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH2
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH3
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM5\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM5\_TRGO
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH2
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH3
  - LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH4
  - LL\_ADC\_INJ\_TRIG\_EXT\_EXTI\_LINE15

**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:**

- CR2 JEXTSEL LL\_ADC\_INJ\_GetTriggerSource
- CR2 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:**

- CR2 JEXTEN LL\_ADC\_INJ\_IsTriggerSourceSWStart

## 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:

- CR2 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).
- On this STM32 serie, group injected sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL\_ADC\_SetSequencersScanMode().
- 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\_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).
- On this STM32 serie, group injected sequencer configuration is conditioned to ADC instance sequencer mode. If ADC instance sequencer mode is disabled, sequencers of all groups (group regular, group injected) can be configured but their execution is disabled (limited to rank 1). Refer to function LL\_ADC\_SetSequencersScanMode().
- 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:

- CR1 DISCEN 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:

- CR1 DISCEN LL\_ADC\_REG\_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
  - 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
  - LL\_ADC\_CHANNEL\_VREFINT (1)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
  - LL\_ADC\_CHANNEL\_VBAT (1)

(1) On STM32F4, parameter available only on ADC instance: ADC1.

(2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

## 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().

## 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
    - 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
    - LL\_ADC\_CHANNEL\_VREFINT (1)
    - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
    - LL\_ADC\_CHANNEL\_VBAT (1)
- (1) On STM32F4, parameter available only on ADC instance: ADC1.
- (2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.
  - (1) 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\_SetSequencerRanks
- JSQR JSQ2 LL\_ADC\_INJ\_SetSequencerRanks
- JSQR JSQ3 LL\_ADC\_INJ\_SetSequencerRanks
- JSQR JSQ4 LL\_ADC\_INJ\_SetSequencerRanks

### 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 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.

## Reference Manual to LL API cross reference:

- CR1 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:

- CR1 JAUTO LL\_ADC\_INJ\_GetTrigAuto

### LL\_ADC\_INJ\_SetOffset

#### Function name

**\_\_STATIC\_INLINE void LL\_ADC\_INJ\_SetOffset (ADC\_TypeDef \* ADCx, uint32\_t Rank, uint32\_t OffsetLevel)**

#### Function description

Set ADC group injected offset.

#### 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
- **OffsetLevel:** Value between Min\_Data=0x000 and Max\_Data=0xFFFF

#### Return values

- **None:**

#### Notes

- It sets: ADC group injected rank to which the offset programmed will be appliedOffset 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.
- Offset cannot be enabled or disabled. To emulate offset disabled, set an offset value equal to 0.

### Reference Manual to LL API cross reference:

- JOFR1 JOFFSET1 LL\_ADC\_INJ\_SetOffset
- JOFR2 JOFFSET2 LL\_ADC\_INJ\_SetOffset
- JOFR3 JOFFSET3 LL\_ADC\_INJ\_SetOffset
- JOFR4 JOFFSET4 LL\_ADC\_INJ\_SetOffset

### LL\_ADC\_INJ\_GetOffset

#### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_ADC\_INJ\_GetOffset (ADC\_TypeDef \* ADCx, uint32\_t Rank)**

#### Function description

Get ADC group injected offset.

### 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=0xFFFF

### Notes

- It gives 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.

### Reference Manual to LL API cross reference:

- JOFR1 JOFFSET1 LL\_ADC\_INJ\_GetOffset
- JOFR2 JOFFSET2 LL\_ADC\_INJ\_GetOffset
- JOFR3 JOFFSET3 LL\_ADC\_INJ\_GetOffset
- JOFR4 JOFFSET4 LL\_ADC\_INJ\_GetOffset

### 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
  - 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
  - LL\_ADC\_CHANNEL\_VREFINT (1)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
  - LL\_ADC\_CHANNEL\_VBAT (1)
- (1) On STM32F4, parameter available only on ADC instance: ADC1.
- (2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.
- **SamplingTime:** This parameter can be one of the following values:
  - LL\_ADC\_SAMPLINGTIME\_3CYCLES
  - LL\_ADC\_SAMPLINGTIME\_15CYCLES
  - LL\_ADC\_SAMPLINGTIME\_28CYCLES
  - LL\_ADC\_SAMPLINGTIME\_56CYCLES
  - LL\_ADC\_SAMPLINGTIME\_84CYCLES
  - LL\_ADC\_SAMPLINGTIME\_112CYCLES
  - LL\_ADC\_SAMPLINGTIME\_144CYCLES
  - LL\_ADC\_SAMPLINGTIME\_480CYCLES

## 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. Refer to reference manual for ADC processing time of this STM32 serie.
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.

**Reference Manual to LL API cross reference:**

- SMPR1 SMP18 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP17 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP16 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP15 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP14 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP13 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP12 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP11 LL\_ADC\_SetChannelSamplingTime
- SMPR1 SMP10 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP9 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP8 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP7 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP6 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP5 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP4 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP3 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP2 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP1 LL\_ADC\_SetChannelSamplingTime
- SMPR2 SMP0 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
    - 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
    - LL\_ADC\_CHANNEL\_VREFINT (1)
    - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
    - LL\_ADC\_CHANNEL\_VBAT (1)
- (1) On STM32F4, parameter available only on ADC instance: ADC1.
- (2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

## Return values

- **Returned:** value can be one of the following values:
  - LL\_ADC\_SAMPLINGTIME\_3CYCLES
  - LL\_ADC\_SAMPLINGTIME\_15CYCLES
  - LL\_ADC\_SAMPLINGTIME\_28CYCLES
  - LL\_ADC\_SAMPLINGTIME\_56CYCLES
  - LL\_ADC\_SAMPLINGTIME\_84CYCLES
  - LL\_ADC\_SAMPLINGTIME\_112CYCLES
  - LL\_ADC\_SAMPLINGTIME\_144CYCLES
  - LL\_ADC\_SAMPLINGTIME\_480CYCLES

## 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. Refer to reference manual for ADC processing time of this STM32 serie.

**Reference Manual to LL API cross reference:**

- SMPR1 SMP18 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP17 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP16 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP15 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP14 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP13 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP12 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP11 LL\_ADC\_GetChannelSamplingTime
- SMPR1 SMP10 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP9 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP8 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP7 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP6 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP5 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP4 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP3 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP2 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP1 LL\_ADC\_GetChannelSamplingTime
- SMPR2 SMP0 LL\_ADC\_GetChannelSamplingTime

**LL\_ADC\_SetAnalogWDMonitChannels**

**Function name**

**`__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDChannelGroup)`**

**Function description**

Set ADC analog watchdog monitored channels: a single channel or all channels, on ADC groups regular and-or injected.

### Parameters

- **ADCx**: ADC instance

- **AWDChannelGroup:** This parameter can be one of the following values:
  - LL\_ADC\_AWD\_DISABLE
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_REG
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_INJ
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG
  - LL\_ADC\_AWD\_CHANNEL\_0\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG
  - LL\_ADC\_AWD\_CHANNEL\_1\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG
  - LL\_ADC\_AWD\_CHANNEL\_2\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG
  - LL\_ADC\_AWD\_CHANNEL\_3\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG
  - LL\_ADC\_AWD\_CHANNEL\_4\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG
  - LL\_ADC\_AWD\_CHANNEL\_5\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG
  - LL\_ADC\_AWD\_CHANNEL\_6\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG
  - LL\_ADC\_AWD\_CHANNEL\_7\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG
  - LL\_ADC\_AWD\_CHANNEL\_8\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG
  - LL\_ADC\_AWD\_CHANNEL\_9\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG
  - LL\_ADC\_AWD\_CHANNEL\_10\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG
  - LL\_ADC\_AWD\_CHANNEL\_11\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG
  - LL\_ADC\_AWD\_CHANNEL\_12\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG
  - LL\_ADC\_AWD\_CHANNEL\_13\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG
  - LL\_ADC\_AWD\_CHANNEL\_14\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_15\_REG
  - LL\_ADC\_AWD\_CHANNEL\_15\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_15\_REG\_INJ



- (2) On devices STM32F42x and STM32F43x, limitation: this internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

#### 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 is only 1 kind 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).

#### Reference Manual to LL API cross reference:

- CR1 AWD1CH LL\_ADC\_SetAnalogWDMonitChannels
- CR1 AWD1SGL LL\_ADC\_SetAnalogWDMonitChannels
- CR1 AWD1EN LL\_ADC\_SetAnalogWDMonitChannels

#### **LL\_ADC\_GetAnalogWDMonitChannels**

#### Function name

`__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx)`

#### Function description

Get ADC analog watchdog monitored channel.

#### Parameters

- **ADCx:** ADC instance

## Return values

- **Returned:** value can be one of the following values:
  - LL\_ADC\_AWD\_DISABLE
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_REG
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_INJ
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG
  - LL\_ADC\_AWD\_CHANNEL\_0\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG
  - LL\_ADC\_AWD\_CHANNEL\_1\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG
  - LL\_ADC\_AWD\_CHANNEL\_2\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG
  - LL\_ADC\_AWD\_CHANNEL\_3\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG
  - LL\_ADC\_AWD\_CHANNEL\_4\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG
  - LL\_ADC\_AWD\_CHANNEL\_5\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG
  - LL\_ADC\_AWD\_CHANNEL\_6\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG
  - LL\_ADC\_AWD\_CHANNEL\_7\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG
  - LL\_ADC\_AWD\_CHANNEL\_8\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG
  - LL\_ADC\_AWD\_CHANNEL\_9\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG
  - LL\_ADC\_AWD\_CHANNEL\_10\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG
  - LL\_ADC\_AWD\_CHANNEL\_11\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG
  - LL\_ADC\_AWD\_CHANNEL\_12\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG
  - LL\_ADC\_AWD\_CHANNEL\_13\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG
  - LL\_ADC\_AWD\_CHANNEL\_14\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_15\_REG
  - LL\_ADC\_AWD\_CHANNEL\_15\_INJ

**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 is only 1 kind 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).

**Reference Manual to LL API cross reference:**

- CR1 AWD1CH LL\_ADC\_GetAnalogWDMonitChannels
- CR1 AWD1SGL LL\_ADC\_GetAnalogWDMonitChannels
- CR1 AWD1EN LL\_ADC\_GetAnalogWDMonitChannels

**LL\_ADC\_SetAnalogWDThresholds**
**Function name**

```
__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)
```

**Function description**

Set ADC analog watchdog threshold value of threshold high or low.

**Parameters**

- **ADCx:** ADC instance
- **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=0xFFFF

**Return values**

- **None:**

**Notes**

- 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 is only 1 kind 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).

**Reference Manual to LL API cross reference:**

- HTR HT LL\_ADC\_SetAnalogWDThresholds
- LTR LT LL\_ADC\_SetAnalogWDThresholds

**LL\_ADC\_GetAnalogWDThresholds**
**Function name**

```
__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDThresholdsHighLow)
```

**Function description**

Get ADC analog watchdog threshold value of threshold high or threshold low.

### Parameters

- **ADCx:** ADC instance
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL\_ADC\_AWD\_THRESHOLD\_HIGH
  - LL\_ADC\_AWD\_THRESHOLD\_LOW

### Return values

- **Value:** between Min\_Data=0x000 and Max\_Data=0xFFFF

### Notes

- 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:

- HTR HT LL\_ADC\_GetAnalogWDThresholds
- LTR LT LL\_ADC\_GetAnalogWDThresholds

### 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
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_SIM\_INJ\_SIM
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_SIM\_INJ\_ALT
  - LL\_ADC\_MULTI\_TRIPLE\_INJ\_SIMULT
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_SIMULT
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_INTERL
  - LL\_ADC\_MULTI\_TRIPLE\_INJ\_ALTERN

#### Return values

- **None:**

#### 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 MULTI 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
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_SIM\_INJ\_SIM
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_SIM\_INJ\_ALT
  - LL\_ADC\_MULTI\_TRIPLE\_INJ\_SIMULT
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_SIMULT
  - LL\_ADC\_MULTI\_TRIPLE\_REG\_INTERL
  - LL\_ADC\_MULTI\_TRIPLE\_INJ\_ALTERN

### 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 MULTI 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_1`
  - `LL_ADC_MULTI_REG_DMA_LIMIT_2`
  - `LL_ADC_MULTI_REG_DMA_LIMIT_3`
  - `LL_ADC_MULTI_REG_DMA_UNLMT_1`
  - `LL_ADC_MULTI_REG_DMA_UNLMT_2`
  - `LL_ADC_MULTI_REG_DMA_UNLMT_3`

### 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()`.

### Reference Manual to LL API cross reference:

- CCR MDMA `LL_ADC_SetMultiDMATransfer`
- CCR DDS `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\_1
  - LL\_ADC\_MULTI\_REG\_DMA\_LIMIT\_2
  - LL\_ADC\_MULTI\_REG\_DMA\_LIMIT\_3
  - LL\_ADC\_MULTI\_REG\_DMA\_UNLMT\_1
  - LL\_ADC\_MULTI\_REG\_DMA\_UNLMT\_2
  - LL\_ADC\_MULTI\_REG\_DMA\_UNLMT\_3

### 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 DDS 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_5CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_14CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_16CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_17CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_18CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_19CYCLES`
  - `LL_ADC_MULTI_TWOSMP_DELAY_20CYCLES`

### 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.

### 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\_5CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_6CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_7CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_8CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_9CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_10CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_11CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_12CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_13CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_14CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_15CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_16CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_17CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_18CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_19CYCLES
  - LL\_ADC\_MULTI\_TWOSMP\_DELAY\_20CYCLES

### Reference Manual to LL API cross reference:

- CCR DELAY LL\_ADC\_GetMultiTwoSamplingDelay

### 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.

### Reference Manual to LL API cross reference:

- CR2 ADON 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:**

**Reference Manual to LL API cross reference:**

- CR2 ADON 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.

**Reference Manual to LL API cross reference:**

- CR2 ADON LL\_ADC\_IsEnabled

**LL\_ADC\_REG\_StartConversionSWStart**
**Function name**

```
__STATIC_INLINE void LL_ADC_REG_StartConversionSWStart (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 only for internal trigger (SW start), not for 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 start must be performed using function LL\_ADC\_REG\_StartConversionExtTrig(). (if external trigger edge would have been set during ADC other settings, ADC conversion would start at trigger event as soon as ADC is enabled).

**Reference Manual to LL API cross reference:**

- CR2 SWSTART LL\_ADC\_REG\_StartConversionSWStart

**LL\_ADC\_REG\_StartConversionExtTrig**
**Function name**

```
__STATIC_INLINE void LL_ADC_REG_StartConversionExtTrig (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)
```

**Function description**

Start ADC group regular conversion from external trigger.

### Parameters

- **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
- **ADCx:** ADC instance

### Return values

- **None:**

### Notes

- ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 serie, this function is relevant for ADC conversion start from external trigger. If internal trigger (SW start) is needed, perform ADC conversion start using function LL\_ADC\_REG\_StartConversionSWStart().

### Reference Manual to LL API cross reference:

- CR2 EXTEN LL\_ADC\_REG\_StartConversionExtTrig

### LL\_ADC\_REG\_StopConversionExtTrig

### Function name

```
__STATIC_INLINE void LL_ADC_REG_StopConversionExtTrig (ADC_TypeDef * ADCx)
```

### Function description

Stop ADC group regular conversion from external trigger.

### Parameters

- **ADCx:** ADC instance

### Return values

- **None:**

### Notes

- No more ADC conversion will start at next trigger event following the ADC stop conversion command. If a conversion is on-going, it will be completed.
- On this STM32 serie, there is no specific command to stop a conversion on-going or to stop ADC converting in continuous mode. These actions can be performed using function LL\_ADC\_Disable().

### Reference Manual to LL API cross reference:

- CR2 EXTEN LL\_ADC\_REG\_StopConversionExtTrig

### 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 DATA1 LL\_ADC\_REG\_ReadMultiConversionData32
- CDR DATA2 LL\_ADC\_REG\_ReadMultiConversionData32

**LL\_ADC\_INJ\_StartConversionSWStart**
**Function name**

```
__STATIC_INLINE void LL_ADC_INJ_StartConversionSWStart (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 only for internal trigger (SW start), not for 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 start must be performed using function LL\_ADC\_INJ\_StartConversionExtTrig(). (if external trigger edge would have been set during ADC other settings, ADC conversion would start at trigger event as soon as ADC is enabled).

**Reference Manual to LL API cross reference:**

- CR2 JSWSTART LL\_ADC\_INJ\_StartConversionSWStart

**LL\_ADC\_INJ\_StartConversionExtTrig**
**Function name**

```
__STATIC_INLINE void LL_ADC_INJ_StartConversionExtTrig (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)
```

**Function description**

Start ADC group injected conversion from external trigger.

**Parameters**

- **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
- **ADCx:** ADC instance

**Return values**

- **None:**

**Notes**

- ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 serie, this function is relevant for ADC conversion start from external trigger. If internal trigger (SW start) is needed, perform ADC conversion start using function LL\_ADC\_INJ\_StartConversionSWStart().

**Reference Manual to LL API cross reference:**

- CR2 JEXTEN LL\_ADC\_INJ\_StartConversionExtTrig

## LL\_ADC\_INJ\_StopConversionExtTrig

### Function name

```
__STATIC_INLINE void LL_ADC_INJ_StopConversionExtTrig (ADC_TypeDef * ADCx)
```

### Function description

Stop ADC group injected conversion from external trigger.

### Parameters

- **ADCx:** ADC instance

### Return values

- **None:**

### Notes

- No more ADC conversion will start at next trigger event following the ADC stop conversion command. If a conversion is on-going, it will be completed.
- On this STM32 serie, there is no specific command to stop a conversion on-going or to stop ADC converting in continuous mode. These actions can be performed using function LL\_ADC\_Disable().

### Reference Manual to LL API cross reference:

- CR2 JEXTEN LL\_ADC\_INJ\_StopConversionExtTrig

## 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 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
- **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=0xFFFF

### 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\_EOCS

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOCS (ADC_TypeDef * ADCx)
```

### Function description

Get flag ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration.

### Parameters

- **ADCx:** ADC instance

### Return values

- **State:** of bit (1 or 0).

### Notes

- To configure flag of end of conversion, use function LL\_ADC\_REG\_SetFlagEndOfConversion().

### Reference Manual to LL API cross reference:

- SR EOC LL\_ADC\_IsActiveFlag\_EOCS

## 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:

- SR OVR LL\_ADC\_IsActiveFlag\_OVR

## 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:

- SR JEOC LL\_ADC\_IsActiveFlag\_JEOS

### 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:

- SR AWD LL\_ADC\_IsActiveFlag\_AWD1

### LL\_ADC\_ClearFlag\_EOCS

#### Function name

```
__STATIC_INLINE void LL_ADC_ClearFlag_EOCS (ADC_TypeDef * ADCx)
```

#### Function description

Clear flag ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration.

#### Parameters

- **ADCx:** ADC instance

#### Return values

- **None:**

#### Notes

- To configure flag of end of conversion, use function LL\_ADC\_REG\_SetFlagEndOfConversion().

#### Reference Manual to LL API cross reference:

- SR EOC LL\_ADC\_ClearFlag\_EOCS

### 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:

- SR OVR LL\_ADC\_ClearFlag\_OVR

## 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:

- SR JEOC LL\_ADC\_ClearFlag\_JEOS

## 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:

- SR AWD LL\_ADC\_ClearFlag\_AWD1

## LL\_ADC\_IsActiveFlag\_MST\_EOCS

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOCS (ADC_Common_TypeDef *
ADCxy_COMMON)
```

### Function description

Get flag multimode ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration, 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).

### Notes

- To configure flag of end of conversion, use function `LL_ADC_REG_SetFlagEndOfConversion()`.

### Reference Manual to LL API cross reference:

- CSR EOC1 LL\_ADC\_IsActiveFlag\_MST\_EOCS

## LL\_ADC\_IsActiveFlag\_SLV1\_EOCS

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV1_EOCS (ADC_Common_TypeDef *
ADCxy_COMMON)
```

### Function description

Get flag multimode ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration, of the ADC slave 1.

### 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).

### Notes

- To configure flag of end of conversion, use function `LL_ADC_REG_SetFlagEndOfConversion()`.

### Reference Manual to LL API cross reference:

- CSR EOC2 LL\_ADC\_IsActiveFlag\_SLV1\_EOCS

## LL\_ADC\_IsActiveFlag\_SLV2\_EOCS

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV2_EOCS (ADC_Common_TypeDef *
ADCxy_COMMON)
```

### Function description

Get flag multimode ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration, of the ADC slave 2.

### 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).

### Notes

- To configure flag of end of conversion, use function `LL_ADC_REG_SetFlagEndOfConversion()`.

### Reference Manual to LL API cross reference:

- CSR EOC3 LL\_ADC\_IsActiveFlag\_SLV2\_EOCS

## 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 OVR1 LL\_ADC\_IsActiveFlag\_MST\_OVR

### LL\_ADC\_IsActiveFlag\_SLV1\_OVR

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV1_OVR (ADC_Common_TypeDef *
ADCxy_COMMON)
```

### Function description

Get flag multimode ADC group regular overrun of the ADC slave 1.

### 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 OVR2 LL\_ADC\_IsActiveFlag\_SLV1\_OVR

### LL\_ADC\_IsActiveFlag\_SLV2\_OVR

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV2_OVR (ADC_Common_TypeDef *
ADCxy_COMMON)
```

### Function description

Get flag multimode ADC group regular overrun of the ADC slave 2.

### 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 OVR3 LL\_ADC\_IsActiveFlag\_SLV2\_OVR

### 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 JEOP LL\_ADC\_IsActiveFlag\_MST\_EOCS

**LL\_ADC\_IsActiveFlag\_SLV1\_JEOS**
**Function name**

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV1_JEOS (ADC_Common_TypeDef *
ADCxy_COMMON)
```

**Function description**

Get flag multimode ADC group injected end of sequence conversions of the ADC slave 1.

**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 JEOP2 LL\_ADC\_IsActiveFlag\_SLV1\_JEOS

**LL\_ADC\_IsActiveFlag\_SLV2\_JEOS**
**Function name**

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV2_JEOS (ADC_Common_TypeDef *
ADCxy_COMMON)
```

**Function description**

Get flag multimode ADC group injected end of sequence conversions of the ADC slave 2.

**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 JEOP3 LL\_ADC\_IsActiveFlag\_SLV2\_JEOS

**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 LL\_ADC\_IsActiveFlag\_MST\_AWD1

### LL\_ADC\_IsActiveFlag\_SLV1\_AWD1

#### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV1_AWD1 (ADC_Common_TypeDef *
ADCxy_COMMON)
```

#### Function description

Get flag multimode analog watchdog 1 of the ADC slave 1.

#### 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 LL\_ADC\_IsActiveFlag\_SLV1\_AWD1

### LL\_ADC\_IsActiveFlag\_SLV2\_AWD1

#### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV2_AWD1 (ADC_Common_TypeDef *
ADCxy_COMMON)
```

#### Function description

Get flag multimode analog watchdog 1 of the ADC slave 2.

#### 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 LL\_ADC\_IsActiveFlag\_SLV2\_AWD1

### LL\_ADC\_EnableIT\_EOCS

#### Function name

```
__STATIC_INLINE void LL_ADC_EnableIT_EOCS (ADC_TypeDef * ADCx)
```

#### Function description

Enable interruption ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration.

#### Parameters

- **ADCx:** ADC instance

#### Return values

- **None:**

#### Notes

- To configure flag of end of conversion, use function `LL_ADC_REG_SetFlagEndOfConversion()`.

#### Reference Manual to LL API cross reference:

- CR1 EOCIE LL\_ADC\_EnableIT\_EOCS



### 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:

- CR1 OVRIE LL\_ADC\_EnableIT\_OVR

### 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:

- CR1 JEOCIE LL\_ADC\_EnableIT\_JEOS

### 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:

- CR1 AWDIE LL\_ADC\_EnableIT\_AWD1

### LL\_ADC\_DisableIT\_EOCS

#### Function name

```
__STATIC_INLINE void LL_ADC_DisableIT_EOCS (ADC_TypeDef * ADCx)
```

**Function description**

Disable interruption ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration.

**Parameters**

- **ADCx:** ADC instance

**Return values**

- **None:**

**Notes**

- To configure flag of end of conversion, use function LL\_ADC\_REG\_SetFlagEndOfConversion().

**Reference Manual to LL API cross reference:**

- CR1 EOCIE LL\_ADC\_DisableIT\_EOCS

**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:**

- CR1 OVRIE LL\_ADC\_DisableIT\_OVR

**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:**

- CR1 JEOCIE LL\_ADC\_EnableIT\_JEOS

**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:

- CR1 AWDIE LL\_ADC\_EnableIT\_AWD1

### LL\_ADC\_IsEnabledIT\_EOCS

### Function name

```
__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOCS (ADC_TypeDef * ADCx)
```

### Function description

Get state of interruption ADC group regular end of unitary conversion or end of sequence conversions, depending on ADC configuration.

### Parameters

- **ADCx:** ADC instance

### Return values

- **State:** of bit (1 or 0).

### Notes

- To configure flag of end of conversion, use function LL\_ADC\_REG\_SetFlagEndOfConversion(). (0: interrupt disabled, 1: interrupt enabled)

### Reference Manual to LL API cross reference:

- CR1 EOCIE LL\_ADC\_IsEnabledIT\_EOCS

### 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:

- CR1 OVRIE LL\_ADC\_IsEnabledIT\_OVR

### 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:

- CR1 JEOCIE LL\_ADC\_EnableIT\_JEOS

### 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:

- CR1 AWDIE LL\_ADC\_EnableIT\_AWD1

### 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

### 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().

### 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:**

## 73.3 ADC Firmware driver defines

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

### 73.3.1 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\_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\_TEMPSENSOR\_REG**

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only. This internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

**LL\_ADC\_AWD\_CH\_TEMPSENSOR\_INJ**

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group injected only. This internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

**LL\_ADC\_AWD\_CH\_TEMPSENSOR\_REG\_INJ**

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by either group regular or injected. This internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

**Analog watchdog - Analog watchdog number**

**LL\_ADC\_AWD1**

ADC analog watchdog number 1

**Analog watchdog - Thresholds**

**LL\_ADC\_AWD\_THRESHOLD\_HIGH**

ADC analog watchdog threshold high

**LL\_ADC\_AWD\_THRESHOLD\_LOW**

ADC analog watchdog threshold low

**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 STM32F4, ADC channel available only on ADC instance: ADC1.

#### 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 STM32F4, ADC channel available only on ADC instance: ADC1.

#### LL\_ADC\_CHANNEL\_TEMPSENSOR

ADC internal channel connected to Temperature sensor. On STM32F4, ADC channel available only on ADC instance: ADC1. This internal channel is shared between temperature sensor and Vbat, only 1 measurement path must be enabled.

##### **Channel - Sampling time**

#### LL\_ADC\_SAMPLINGTIME\_3CYCLES

Sampling time 3 ADC clock cycles

#### LL\_ADC\_SAMPLINGTIME\_15CYCLES

Sampling time 15 ADC clock cycles

#### LL\_ADC\_SAMPLINGTIME\_28CYCLES

Sampling time 28 ADC clock cycles

#### LL\_ADC\_SAMPLINGTIME\_56CYCLES

Sampling time 56 ADC clock cycles

#### LL\_ADC\_SAMPLINGTIME\_84CYCLES

Sampling time 84 ADC clock cycles

**LL\_ADC\_SAMPLINGTIME\_112CYCLES**

Sampling time 112 ADC clock cycles

**LL\_ADC\_SAMPLINGTIME\_144CYCLES**

Sampling time 144 ADC clock cycles

**LL\_ADC\_SAMPLINGTIME\_480CYCLES**

Sampling time 480 ADC clock cycles

**ADC common - Clock source**

**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\_SYNC\_PCLK\_DIV6**

ADC synchronous clock derived from AHB clock with prescaler division by 6

**LL\_ADC\_CLOCK\_SYNC\_PCLK\_DIV8**

ADC synchronous clock derived from AHB clock with prescaler division by 8

**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 (alignment on data register MSB bit 15)

**ADC flags**

**LL\_ADC\_FLAG\_STRT**

ADC flag ADC group regular conversion start

**LL\_ADC\_FLAG\_EOCS**

ADC flag ADC group regular end of unitary conversion or sequence conversions (to configure flag of end of conversion, use function

**LL\_ADC\_FLAG\_OVR**

ADC flag ADC group regular overrun

**LL\_ADC\_FLAG\_JSTRT**

ADC flag ADC group injected conversion start

**LL\_ADC\_FLAG\_JEOS**

ADC flag ADC group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)

**LL\_ADC\_FLAG\_AWD1**

ADC flag ADC analog watchdog 1

**LL\_ADC\_FLAG\_EOCS\_MST**

ADC flag ADC multimode master group regular end of unitary conversion or sequence conversions (to configure flag of end of conversion, use function

**LL\_ADC\_FLAG\_EOCS\_SLV1**

ADC flag ADC multimode slave 1 group regular end of unitary conversion or sequence conversions (to configure flag of end of conversion, use function

**LL\_ADC\_FLAG\_EOCS\_SLV2**

ADC flag ADC multimode slave 2 group regular end of unitary conversion or sequence conversions (to configure flag of end of conversion, use function

**LL\_ADC\_FLAG\_OVR\_MST**

ADC flag ADC multimode master group regular overrun

**LL\_ADC\_FLAG\_OVR\_SLV1**

ADC flag ADC multimode slave 1 group regular overrun

**LL\_ADC\_FLAG\_OVR\_SLV2**

ADC flag ADC multimode slave 2 group regular overrun

**LL\_ADC\_FLAG\_JEOS\_MST**

ADC flag ADC multimode master group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)

**LL\_ADC\_FLAG\_JEOS\_SLV1**

ADC flag ADC multimode slave 1 group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)

**LL\_ADC\_FLAG\_JEOS\_SLV2**

ADC flag ADC multimode slave 2 group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)

**LL\_ADC\_FLAG\_AWD1\_MST**

ADC flag ADC multimode master analog watchdog 1 of the ADC master

**LL\_ADC\_FLAG\_AWD1\_SLV1**

ADC flag ADC multimode slave 1 analog watchdog 1

**LL\_ADC\_FLAG\_AWD1\_SLV2**

ADC flag ADC multimode slave 2 analog watchdog 1

**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\_VREFINT\_STAB\_US

Delay for internal voltage reference stabilization time

#### LL\_ADC\_DELAY\_TEMPSENSOR\_STAB\_US

Delay for internal voltage reference stabilization time

**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.

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM1\_CH4**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM1\_TRGO**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM2\_CH1**

ADC group injected conversion trigger from external IP: TIM2 channel 1 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 IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM3\_CH2**

ADC group injected conversion trigger from external IP: TIM3 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM3\_CH4**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH1**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH2**

ADC group injected conversion trigger from external IP: TIM4 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_CH3**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM4\_TRGO**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM5\_CH4**

ADC group injected conversion trigger from external IP: TIM5 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM5\_TRGO**

ADC group injected conversion trigger from external IP: TIM5 TRGO. Trigger edge set to rising edge (default setting).

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH2**

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

**LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH3**

ADC group injected conversion trigger from external IP: TIM8 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

#### LL\_ADC\_INJ\_TRIG\_EXT\_TIM8\_CH4

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

#### LL\_ADC\_INJ\_TRIG\_EXT\_EXTI\_LINE15

ADC group injected conversion trigger from external IP: external interrupt line 15. 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\_EOCS

ADC interruption ADC group regular end of unitary conversion or sequence conversions (to configure flag of end of conversion, use function

#### LL\_ADC\_IT\_OVR

ADC interruption ADC group regular overrun

#### LL\_ADC\_IT\_JEOS

ADC interruption ADC group injected end of sequence conversions (Note: on this STM32 serie, there is no flag ADC group injected end of unitary conversion. Flag noted as "JEOC" is corresponding to flag "JEOS" in other STM32 families)

#### LL\_ADC\_IT\_AWD1

ADC interruption ADC analog watchdog 1

**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\_1

ADC multimode group regular conversions are transferred by DMA, one DMA channel for all ADC instances (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 of DMA mode 1: 2 or 3 (dual or triple mode) half-words one by one, ADC1 then ADC2 then ADC3.

#### LL\_ADC\_MULTI\_REG\_DMA\_LIMIT\_2

ADC multimode group regular conversions are transferred by DMA, one DMA channel for all ADC instances (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 of DMA mode 2: 2 or 3 (dual or triple mode) half-words one by one, ADC2&1 then ADC1&3 then ADC3&2.

#### LL\_ADC\_MULTI\_REG\_DMA\_LIMIT\_3

ADC multimode group regular conversions are transferred by DMA, one DMA channel for all ADC instances (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 of DMA mode 3: 2 or 3 (dual or triple mode) bytes one by one, ADC2&1 then ADC1&3 then ADC3&2.

**LL\_ADC\_MULTI\_REG\_DMA\_UNLMT\_1**

ADC multimode group regular conversions are transferred by DMA, one DMA channel for all ADC instances (DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting of DMA mode 1: 2 or 3 (dual or triple mode) half-words one by one, ADC1 then ADC2 then ADC3.

**LL\_ADC\_MULTI\_REG\_DMA\_UNLMT\_2**

ADC multimode group regular conversions are transferred by DMA, one DMA channel for all ADC instances (DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting of DMA mode 2: 2 or 3 (dual or triple mode) half-words by pairs, ADC2&1 then ADC1&3 then ADC3&2.

**LL\_ADC\_MULTI\_REG\_DMA\_UNLMT\_3**

ADC multimode group regular conversions are transferred by DMA, one DMA channel for all ADC instances (DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting of DMA mode 3: 2 or 3 (dual or triple mode) bytes one by one, ADC2&1 then ADC1&3 then ADC3&2.

**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

**LL\_ADC\_MULTI\_TRIPLE\_REG\_SIM\_INJ\_SIM**

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

**LL\_ADC\_MULTI\_TRIPLE\_REG\_SIM\_INJ\_ALT**

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

**LL\_ADC\_MULTI\_TRIPLE\_INJ\_SIMULT**

ADC triple mode enabled: group injected simultaneous

**LL\_ADC\_MULTI\_TRIPLE\_REG\_SIMULT**

ADC triple mode enabled: group regular simultaneous

**LL\_ADC\_MULTI\_TRIPLE\_REG\_INTERL**

ADC triple mode enabled: Combined group regular interleaved

**LL\_ADC\_MULTI\_TRIPLE\_INJ\_ALTERN**

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

***Multimode - Delay between two sampling phases***

**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

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_13CYCLES**

ADC multimode delay between two sampling phases: 13 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_14CYCLES**

ADC multimode delay between two sampling phases: 14 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_15CYCLES**

ADC multimode delay between two sampling phases: 15 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_16CYCLES**

ADC multimode delay between two sampling phases: 16 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_17CYCLES**

ADC multimode delay between two sampling phases: 17 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_18CYCLES**

ADC multimode delay between two sampling phases: 18 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_19CYCLES**

ADC multimode delay between two sampling phases: 19 ADC clock cycles

**LL\_ADC\_MULTI\_TWOSMP\_DELAY\_20CYCLES**

ADC multimode delay between two sampling phases: 20 ADC clock cycles

**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 - Flag EOC selection (unitary or sequence conversions)**

**LL\_ADC\_REG\_FLAG\_EOC\_SEQUENCE\_CONV**

ADC flag EOC (end of unitary conversion) selected

**LL\_ADC\_REG\_FLAG\_EOC\_UNITARY\_CONV**

ADC flag EOS (end of sequence conversions) selected

**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\_CH1**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM1\_CH2**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM1\_CH3**

ADC group regular conversion trigger from external IP: 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\_CH2**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH3**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM2\_CH4**

ADC group regular conversion trigger from external IP: TIM2 channel 4 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 IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).

**LL\_ADC\_REG\_TRIG\_EXT\_TIM3\_CH1**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM3\_TRGO**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM4\_CH4**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH1**

ADC group regular conversion trigger from external IP: TIM5 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

**LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH2**

ADC group regular conversion trigger from external IP: TIM5 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

**LL\_ADC\_REG\_TRIG\_EXT\_TIM5\_CH3**

ADC group regular conversion trigger from external IP: TIM5 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

**LL\_ADC\_REG\_TRIG\_EXT\_TIM8\_CH1**

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

**LL\_ADC\_REG\_TRIG\_EXT\_TIM8\_TRGO**

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

**LL\_ADC\_REG\_TRIG\_EXT\_EXTI\_LINE11**

ADC group regular conversion trigger from external IP: external interrupt line 11. 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 - Scan selection***
**LL\_ADC\_SEQ\_SCAN\_DISABLE**

ADC conversion is performed in unitary conversion mode (one channel converted, that defined in rank 1). Configuration of both groups regular and injected sequencers (sequence length, ...) is discarded: equivalent to length of 1 rank.

**LL\_ADC\_SEQ\_SCAN\_ENABLE**

ADC conversions are performed in sequence conversions mode, according to configuration of both groups regular and injected sequencers (sequence length, ...).

***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`
  - `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`
  - `LL_ADC_CHANNEL_VREFINT` (1)
  - `LL_ADC_CHANNEL_TEMPSENSOR` (1)(2)
  - `LL_ADC_CHANNEL_VBAT` (1)

### 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`
  - `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`
  - `LL_ADC_CHANNEL_VREFINT` (1)
  - `LL_ADC_CHANNEL_TEMPSENSOR` (1)(2)
  - `LL_ADC_CHANNEL_VBAT` (1)

**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
  - 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
  - LL\_ADC\_CHANNEL\_VREFINT (1)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
  - LL\_ADC\_CHANNEL\_VBAT (1)

**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
  - 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
  - LL\_ADC\_CHANNEL\_VREFINT (1)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
  - LL\_ADC\_CHANNEL\_VBAT (1)

### 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 (1)
  - LL\_ADC\_CHANNEL\_TEMPSENSOR (1)(2)
  - LL\_ADC\_CHANNEL\_VBAT (1)

**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`
  - `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`
  - `LL_ADC_CHANNEL_VREFINT (1)`
  - `LL_ADC_CHANNEL_TEMPSENSOR (1)(2)`
  - `LL_ADC_CHANNEL_VBAT (1)`
- `__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
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_INJ
  - LL\_ADC\_AWD\_ALL\_CHANNELS\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG
  - LL\_ADC\_AWD\_CHANNEL\_0\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_0\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG
  - LL\_ADC\_AWD\_CHANNEL\_1\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_1\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG
  - LL\_ADC\_AWD\_CHANNEL\_2\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_2\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG
  - LL\_ADC\_AWD\_CHANNEL\_3\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_3\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG
  - LL\_ADC\_AWD\_CHANNEL\_4\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_4\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG
  - LL\_ADC\_AWD\_CHANNEL\_5\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_5\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG
  - LL\_ADC\_AWD\_CHANNEL\_6\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_6\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG
  - LL\_ADC\_AWD\_CHANNEL\_7\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_7\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG
  - LL\_ADC\_AWD\_CHANNEL\_8\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_8\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG
  - LL\_ADC\_AWD\_CHANNEL\_9\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_9\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG
  - LL\_ADC\_AWD\_CHANNEL\_10\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_10\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG
  - LL\_ADC\_AWD\_CHANNEL\_11\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_11\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG
  - LL\_ADC\_AWD\_CHANNEL\_12\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_12\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG
  - LL\_ADC\_AWD\_CHANNEL\_13\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_13\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG
  - LL\_ADC\_AWD\_CHANNEL\_14\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_14\_REG\_INJ
  - LL\_ADC\_AWD\_CHANNEL\_15\_REG
  - LL\_ADC\_AWD\_CHANNEL\_15\_INJ

**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_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\_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=0xFFFF

**Return value:**

- Value: between Min\_Data=0x000 and Max\_Data=0xFFFF

**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 be processed with this macro to separate the conversion data of ADC master and ADC slave.

### **\_\_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 instances Multimode (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 equivalent voltage value (unit: mVolt)

**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\_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 STM32F4, 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 STM32F4, refer to device datasheet parameter "V25".
- **\_\_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 12bits (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

## 74 LL BUS Generic Driver

### 74.1 BUS Firmware driver API description

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

#### 74.1.1 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\_GPIOA
- LL\_AHB1\_GRP1\_PERIPH\_GPIOB
- LL\_AHB1\_GRP1\_PERIPH\_GPIOC
- LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
- LL\_AHB1\_GRP1\_PERIPH\_CRC
- LL\_AHB1\_GRP1\_PERIPH\_BKPSRAM (\*)
- LL\_AHB1\_GRP1\_PERIPH\_CCMDATARAM (\*)
- LL\_AHB1\_GRP1\_PERIPH\_DMA1
- LL\_AHB1\_GRP1\_PERIPH\_DMA2
- LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
- LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMACTX (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMACRX (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMACPTP (\*)
- LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)
- LL\_AHB1\_GRP1\_PERIPH\_OTGHSULPI (\*)

(\*) value not defined in all devices.

###### Return values

- **None:**



**Reference Manual to LL API cross reference:**

- AHB1ENR GPIOAEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOBEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOCEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIODEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOEEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOFEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOGEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOHEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOIEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOJEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR GPIOKEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR CRCEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR BKPSRAMEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR CCMDATARAMEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR DMA1EN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR DMA2EN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR RNGEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR DMA2DEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR ETHMACEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR ETHMACTXEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR ETHMACRXEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR ETHMACPTPEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR OTGHSSEN LL\_AHB1\_GRP1\_EnableClock
- AHB1ENR OTGHSULPIEN 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\_GPIOA
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOB
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOC
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CRC
    - LL\_AHB1\_GRP1\_PERIPH\_BKPSRAM (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CCMDATARAM (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_DMA1
    - LL\_AHB1\_GRP1\_PERIPH\_DMA2
    - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMACTX (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMACRX (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMACPTP (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_OTGHSULPI (\*)
- (\*) value not defined in all devices.

## Return values

- **State:** of Periphs (1 or 0).

**Reference Manual to LL API cross reference:**

- AHB1ENR GPIOAEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOBEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOCEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIODEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOEEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOFEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOGEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOHEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOIEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOJEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR GPIOKEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR CRCEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR BKPSRAMEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR CCMDATARAMEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR DMA1EN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR DMA2EN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR RNGEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR DMA2DEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR ETHMACEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR ETHMACTXEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR ETHMACRXEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR ETHMACPTPEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR OTGHSSEN LL\_AHB1\_GRP1\_IsEnabledClock
- AHB1ENR OTGHSULPIEN 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\_GPIOA
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOB
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOC
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CRC
    - LL\_AHB1\_GRP1\_PERIPH\_BKPSRAM (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CCMDATARAM (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_DMA1
    - LL\_AHB1\_GRP1\_PERIPH\_DMA2
    - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMACTX (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMACRX (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMACPTP (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_OTGHSULPI (\*)
- (\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- AHB1ENR GPIOAEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOBEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOCEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIODEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOEEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOFEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOGEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOHEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOIEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOJEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR GPIOKEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR CRCEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR BKPSRAMEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR CCMDATARAMEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR DMA1EN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR DMA2EN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR RNGEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR DMA2DEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR ETHMACEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR ETHMACTXEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR ETHMACRXEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR ETHMACPTPEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR OTGHSSEN LL\_AHB1\_GRP1\_DisableClock
- AHB1ENR OTGHSULPIEN 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\_GPIOA
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOB
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOC
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_CRC
  - LL\_AHB1\_GRP1\_PERIPH\_DMA1
  - LL\_AHB1\_GRP1\_PERIPH\_DMA2
  - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- AHB1RSTR GPIOARST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOBRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOCRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIODRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOERST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOFRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOGRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOHRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOIRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOJRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR GPIOKRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR CRCRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR DMA1RST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR DMA2RST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR RNGRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR DMA2DRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR ETHMACRST LL\_AHB1\_GRP1\_ForceReset
- AHB1RSTR OTGHSRST 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\_GPIOA
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOB
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOC
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_CRC
    - LL\_AHB1\_GRP1\_PERIPH\_DMA1
    - LL\_AHB1\_GRP1\_PERIPH\_DMA2
    - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
    - LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)
- (\*) value not defined in all devices.

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- AHB1RSTR GPIOARST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOBRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOCRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIODRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOERST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOFRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOGRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOHRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOIRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOJRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR GPIOKRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR CRCRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR DMA1RST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR DMA2RST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR RNGRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR DMA2DRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR ETHMACRST LL\_AHB1\_GRP1\_ReleaseReset
- AHB1RSTR OTGHSRST LL\_AHB1\_GRP1\_ReleaseReset

## LL\_AHB1\_GRP1\_EnableClockLowPower

### Function name

`__STATIC_INLINE void LL_AHB1_GRP1_EnableClockLowPower (uint32_t Periphs)`

### Function description

Enable AHB1 peripheral clocks in low-power mode.

### Parameters

- **Periphs:** This parameter can be a combination of the following values:

- LL\_AHB1\_GRP1\_PERIPH\_GPIOA
- LL\_AHB1\_GRP1\_PERIPH\_GPIOB
- LL\_AHB1\_GRP1\_PERIPH\_GPIOC
- LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
- LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
- LL\_AHB1\_GRP1\_PERIPH\_CRC
- LL\_AHB1\_GRP1\_PERIPH\_BKPSRAM (\*)
- LL\_AHB1\_GRP1\_PERIPH\_FLITF
- LL\_AHB1\_GRP1\_PERIPH\_SRAM1
- LL\_AHB1\_GRP1\_PERIPH\_SRAM2 (\*)
- LL\_AHB1\_GRP1\_PERIPH\_SRAM3 (\*)
- LL\_AHB1\_GRP1\_PERIPH\_DMA1
- LL\_AHB1\_GRP1\_PERIPH\_DMA2
- LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
- LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMACTX (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMACRX (\*)
- LL\_AHB1\_GRP1\_PERIPH\_ETHMACPTP (\*)
- LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)
- LL\_AHB1\_GRP1\_PERIPH\_OTGHSULPI (\*)

(\*) value not defined in all devices.

### Return values

- **None:**



**Reference Manual to LL API cross reference:**

- AHB1LPENR GPIOALPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOBLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOCLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIODLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOELPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOFLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOGLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOHLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOILPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOJLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR GPIOKLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR CRCLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR BKPSRAMLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR FLITFLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR SRAM1LPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR SRAM2LPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR SRAM3LPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR BKPSRAMLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR DMA1LPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR DMA2LPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR DMA2DLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR RNGLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR ETHMACLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR ETHMACTXLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR ETHMACRXLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR ETHMACPTLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR OTGHSLPEN LL\_AHB1\_GRP1\_EnableClockLowPower
- AHB1LPENR OTGHSULPILPEN LL\_AHB1\_GRP1\_EnableClockLowPower

**LL\_AHB1\_GRP1\_DisableClockLowPower**
**Function name**
**`__STATIC_INLINE void LL_AHB1_GRP1_DisableClockLowPower (uint32_t Periph)`**
**Function description**

Disable AHB1 peripheral clocks in low-power mode.

## Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOA
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOB
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOC
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOD (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOE (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOF (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOG (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOH (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOI (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOJ (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_GPIOK (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_CRC
  - LL\_AHB1\_GRP1\_PERIPH\_BKPSRAM (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_FLITF
  - LL\_AHB1\_GRP1\_PERIPH\_SRAM1
  - LL\_AHB1\_GRP1\_PERIPH\_SRAM2 (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_SRAM3 (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_DMA1
  - LL\_AHB1\_GRP1\_PERIPH\_DMA2
  - LL\_AHB1\_GRP1\_PERIPH\_RNG (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_DMA2D (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_ETHMAC (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_ETHMACTX (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_ETHMACRX (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_ETHMACPTP (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_OTGHS (\*)
  - LL\_AHB1\_GRP1\_PERIPH\_OTGHSULPI (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- AHB1LPENR GPIOALPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOBLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOCLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIODLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOELPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOFLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOGLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOHLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOILPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOJLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR GPIOKLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR CRCLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR BKPSRAMLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR FLITFLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR SRAM1LPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR SRAM2LPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR SRAM3LPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR BKPSRAMLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR DMA1LPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR DMA2LPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR DMA2DLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR RNGLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR ETHMACLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR ETHMACTXLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR ETHMACRXLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR ETHMACPTLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR OTGHSLPEN LL\_AHB1\_GRP1\_DisableClockLowPower
- AHB1LPENR OTGHSULPILPEN LL\_AHB1\_GRP1\_DisableClockLowPower

**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\_DCMI (\*)
  - LL\_AHB2\_GRP1\_PERIPH\_Cryp (\*)
  - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
  - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
  - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
  - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)

(\*) value not defined in all devices.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- AHB2ENR DCMIEN LL\_AHB2\_GRP1\_EnableClock
- AHB2ENR CRYPEN LL\_AHB2\_GRP1\_EnableClock
- AHB2ENR AESEN LL\_AHB2\_GRP1\_EnableClock
- AHB2ENR HASHEN LL\_AHB2\_GRP1\_EnableClock
- AHB2ENR RNGEN LL\_AHB2\_GRP1\_EnableClock
- AHB2ENR OTGFSEN 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\_DCMI (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_CRYP (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)
- (\*) value not defined in all devices.

**Return values**

- **State:** of Periphs (1 or 0).

**Reference Manual to LL API cross reference:**

- AHB2ENR DCMIEN LL\_AHB2\_GRP1\_IsEnabledClock
- AHB2ENR CRYPEN LL\_AHB2\_GRP1\_IsEnabledClock
- AHB2ENR AESEN LL\_AHB2\_GRP1\_IsEnabledClock
- AHB2ENR HASHEN LL\_AHB2\_GRP1\_IsEnabledClock
- AHB2ENR RNGEN LL\_AHB2\_GRP1\_IsEnabledClock
- AHB2ENR OTGFSEN 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\_DCMI (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_CRYP (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)
- (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL\_AHB2\_GRP1\_DisableClock
- AHB2ENR CRYPEN LL\_AHB2\_GRP1\_DisableClock
- AHB2ENR AESEN LL\_AHB2\_GRP1\_DisableClock
- AHB2ENR HASHEN LL\_AHB2\_GRP1\_DisableClock
- AHB2ENR RNGEN LL\_AHB2\_GRP1\_DisableClock
- AHB2ENR OTGFSEN 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\_ALL
    - LL\_AHB2\_GRP1\_PERIPH\_DCMI (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_CRYP (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)
- (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB2RSTR DCMIRST LL\_AHB2\_GRP1\_ForceReset
- AHB2RSTR CRYPREST LL\_AHB2\_GRP1\_ForceReset
- AHB2RSTR AESRST LL\_AHB2\_GRP1\_ForceReset
- AHB2RSTR HASHRST LL\_AHB2\_GRP1\_ForceReset
- AHB2RSTR RNGRST LL\_AHB2\_GRP1\_ForceReset
- AHB2RSTR OTGFSRST 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\_ALL
    - LL\_AHB2\_GRP1\_PERIPH\_DCMI (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_Cryp (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)
- (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB2RSTR DCMIRST LL\_AHB2\_GRP1\_ReleaseReset
- AHB2RSTR CRYPYST LL\_AHB2\_GRP1\_ReleaseReset
- AHB2RSTR AESRST LL\_AHB2\_GRP1\_ReleaseReset
- AHB2RSTR HASHRST LL\_AHB2\_GRP1\_ReleaseReset
- AHB2RSTR RNGRST LL\_AHB2\_GRP1\_ReleaseReset
- AHB2RSTR OTGFSRST LL\_AHB2\_GRP1\_ReleaseReset

### LL\_AHB2\_GRP1\_EnableClockLowPower

#### Function name

`__STATIC_INLINE void LL_AHB2_GRP1_EnableClockLowPower (uint32_t Periphs)`

#### Function description

Enable AHB2 peripheral clocks in low-power mode.

### Parameters

- **Periphs:** This parameter can be a combination of the following values:
    - LL\_AHB2\_GRP1\_PERIPH\_DCMI (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_Cryp (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)
- (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL\_AHB2\_GRP1\_EnableClockLowPower
- AHB2LPENR CRYPYPEN LL\_AHB2\_GRP1\_EnableClockLowPower
- AHB2LPENR AESLPEN LL\_AHB2\_GRP1\_EnableClockLowPower
- AHB2LPENR HASHLPEN LL\_AHB2\_GRP1\_EnableClockLowPower
- AHB2LPENR RNLLEN LL\_AHB2\_GRP1\_EnableClockLowPower
- AHB2LPENR OTGFSLPEN LL\_AHB2\_GRP1\_EnableClockLowPower

## LL\_AHB2\_GRP1\_DisableClockLowPower

### Function name

```
__STATIC_INLINE void LL_AHB2_GRP1_DisableClockLowPower (uint32_t Periphs)
```

### Function description

Disable AHB2 peripheral clocks in low-power mode.

### Parameters

- **Periphs:** This parameter can be a combination of the following values:
    - LL\_AHB2\_GRP1\_PERIPH\_DCMI (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_Cryp (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_AES (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_HASH (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_RNG (\*)
    - LL\_AHB2\_GRP1\_PERIPH\_OTGFS (\*)
- (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL\_AHB2\_GRP1\_DisableClockLowPower
- AHB2LPENR CRYPLPEN LL\_AHB2\_GRP1\_DisableClockLowPower
- AHB2LPENR AESLPEN LL\_AHB2\_GRP1\_DisableClockLowPower
- AHB2LPENR HASHLPEN LL\_AHB2\_GRP1\_DisableClockLowPower
- AHB2LPENR RNLGLEN LL\_AHB2\_GRP1\_DisableClockLowPower
- AHB2LPENR OTGFSLPEN LL\_AHB2\_GRP1\_DisableClockLowPower

## 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\_FSMC (\*)
    - 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 FSMCEN 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\_FSMC (\*)
  - 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 FSMCEN 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\_FSMC (\*)
  - 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 FSMCEN 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\_FSMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_QSPI (\*)
 (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB3RSTR FMC RST LL\_AHB3\_GRP1\_ForceReset
- AHB3RSTR FSMC RST LL\_AHB3\_GRP1\_ForceReset
- AHB3RSTR QSPI RST 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\_AHB2\_GRP1\_PERIPH\_ALL
  - LL\_AHB3\_GRP1\_PERIPH\_FMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_FSMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_QSPI (\*)
 (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AHB3RSTR FMC RST LL\_AHB3\_GRP1\_ReleaseReset
- AHB3RSTR FSMC RST LL\_AHB3\_GRP1\_ReleaseReset
- AHB3RSTR QSPI RST LL\_AHB3\_GRP1\_ReleaseReset

### LL\_AHB3\_GRP1\_EnableClockLowPower

#### Function name

`__STATIC_INLINE void LL_AHB3_GRP1_EnableClockLowPower (uint32_t Periphs)`

#### Function description

Enable AHB3 peripheral clocks in low-power mode.

### Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL\_AHB3\_GRP1\_PERIPH\_FMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_FSMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_QSPI (\*)
 (\*) value not defined in all devices.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- AHB3LPENR FMCLPEN LL\_AHB3\_GRP1\_EnableClockLowPower
- AHB3LPENR FSMCLPEN LL\_AHB3\_GRP1\_EnableClockLowPower
- AHB3LPENR QSPILPEN LL\_AHB3\_GRP1\_EnableClockLowPower

#### LL\_AHB3\_GRP1\_DisableClockLowPower

#### Function name

**\_\_STATIC\_INLINE void LL\_AHB3\_GRP1\_DisableClockLowPower (uint32\_t Periphs)**

#### Function description

Disable AHB3 peripheral clocks in low-power mode.

#### Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL\_AHB3\_GRP1\_PERIPH\_FMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_FSMC (\*)
  - LL\_AHB3\_GRP1\_PERIPH\_QSPI (\*)
 (\*) value not defined in all devices.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- AHB3LPENR FMCLPEN LL\_AHB3\_GRP1\_DisableClockLowPower
- AHB3LPENR FSMCLPEN LL\_AHB3\_GRP1\_DisableClockLowPower
- AHB3LPENR QSPILPEN LL\_AHB3\_GRP1\_DisableClockLowPower

#### 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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)
- LL\_APB1\_GRP1\_PERIPH\_RTCAPB (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB1ENR TIM2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM4EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM5EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM6EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM7EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM12EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM13EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR TIM14EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR LPTIM1EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR WWDGEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR SPI2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR SPI3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR SPDIFRXEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR USART2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR USART3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR UART4EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR UART5EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR I2C1EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR I2C2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR I2C3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR FMPI2C1EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR CAN1EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR CAN2EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR CAN3EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR CECEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR PWREN LL\_APB1\_GRP1\_EnableClock
- APB1ENR DACEN LL\_APB1\_GRP1\_EnableClock
- APB1ENR UART7EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR UART8EN LL\_APB1\_GRP1\_EnableClock
- APB1ENR RTCAPBEN LL\_APB1\_GRP1\_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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)
- LL\_APB1\_GRP1\_PERIPH\_RTCAPB (\*)

(\*) value not defined in all devices.

## Return values

- **State:** of Periphs (1 or 0).

**Reference Manual to LL API cross reference:**

- APB1ENR TIM2EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM3EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM4EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM5EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM6EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM7EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM12EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM13EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR TIM14EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR LPTIM1EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR WWDGEN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR SPI2EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR SPI3EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR SPDIFRXEN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR USART2EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR USART3EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR UART4EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR UART5EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR I2C1EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR I2C2EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR I2C3EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR FMPI2C1EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR CAN1EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR CAN2EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR CAN3EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR CECEN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR PWREN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR DACEN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR UART7EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR UART8EN LL\_APB1\_GRP1\_IsEnabledClock
- APB1ENR RTCAPBEN LL\_APB1\_GRP1\_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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)
- LL\_APB1\_GRP1\_PERIPH\_RTCAPB (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB1ENR TIM2EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM3EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM4EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM5EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM6EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM7EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM12EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM13EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR TIM14EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR LPTIM1EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR WWDGEN LL\_APB1\_GRP1\_DisableClock
- APB1ENR SPI2EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR SPI3EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR SPDIFRXEN LL\_APB1\_GRP1\_DisableClock
- APB1ENR USART2EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR USART3EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR UART4EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR UART5EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR I2C1EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR I2C2EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR I2C3EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR FMPI2C1EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR CAN1EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR CAN2EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR CAN3EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR CECEN LL\_APB1\_GRP1\_DisableClock
- APB1ENR PWREN LL\_APB1\_GRP1\_DisableClock
- APB1ENR DACEN LL\_APB1\_GRP1\_DisableClock
- APB1ENR UART7EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR UART8EN LL\_APB1\_GRP1\_DisableClock
- APB1ENR RTCAPBEN LL\_APB1\_GRP1\_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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB1RSTR TIM2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM3RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM4RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM5RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM6RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM7RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM12RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM13RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR TIM14RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR LPTIM1RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR WWDGRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR SPI2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR SPI3RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR SPDIFRXRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR USART2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR USART3RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR UART4RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR UART5RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR I2C1RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR I2C2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR I2C3RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR FMPI2C1RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR CAN1RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR CAN2RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR CAN3RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR CECRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR PWRRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR DACRST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR UART7RST LL\_APB1\_GRP1\_ForceReset
- APB1RSTR UART8RST LL\_APB1\_GRP1\_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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB1RSTR TIM2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM3RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM4RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM5RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM6RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM7RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM12RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM13RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR TIM14RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR LPTIM1RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR WWDGRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR SPI2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR SPI3RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR SPDIFRXRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR USART2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR USART3RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR UART4RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR UART5RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR I2C1RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR I2C2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR I2C3RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR FMPI2C1RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR CAN1RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR CAN2RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR CAN3RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR CECRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR PWRRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR DACRST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR UART7RST LL\_APB1\_GRP1\_ReleaseReset
- APB1RSTR UART8RST LL\_APB1\_GRP1\_ReleaseReset

**LL\_APB1\_GRP1\_EnableClockLowPower**
**Function name**

```
__STATIC_INLINE void LL_APB1_GRP1_EnableClockLowPower (uint32_t Periphs)
```

**Function description**

Enable APB1 peripheral clocks in low-power mode.

## 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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)
- LL\_APB1\_GRP1\_PERIPH\_RTCAPB (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB1LPENR TIM2LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM3LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM4LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM5LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM6LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM7LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM12LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM13LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR TIM14LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR LPTIM1LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR WWDGLPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR SPI2LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR SPI3LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR SPDIFRXPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR USART2LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR USART3LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR UART4LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR UART5LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR I2C1LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR I2C2LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR I2C3LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR FMPI2C1LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR CAN1LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR CAN2LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR CAN3LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR CECLPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR PWRLPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR DAACLPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR UART7LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR UART8LPEN LL\_APB1\_GRP1\_EnableClockLowPower
- APB1LPENR RTCAPBLPEN LL\_APB1\_GRP1\_EnableClockLowPower

**LL\_APB1\_GRP1\_DisableClockLowPower**
**Function name**
**`__STATIC_INLINE void LL_APB1_GRP1_DisableClockLowPower (uint32_t Periphs)`**
**Function description**

Disable APB1 peripheral clocks in low-power mode.

## 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\_TIM12 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM13 (\*)
- LL\_APB1\_GRP1\_PERIPH\_TIM14 (\*)
- LL\_APB1\_GRP1\_PERIPH\_LPTIM1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_WWDG
- LL\_APB1\_GRP1\_PERIPH\_SPI2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPI3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_SPDIFRX (\*)
- 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\_I2C3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_FMPI2C1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN2 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CAN3 (\*)
- LL\_APB1\_GRP1\_PERIPH\_CEC (\*)
- LL\_APB1\_GRP1\_PERIPH\_PWR
- LL\_APB1\_GRP1\_PERIPH\_DAC1 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART7 (\*)
- LL\_APB1\_GRP1\_PERIPH\_UART8 (\*)
- LL\_APB1\_GRP1\_PERIPH\_RTCAPB (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB1LPENR TIM2LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM3LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM4LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM5LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM6LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM7LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM12LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM13LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR TIM14LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR LPTIM1LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR WWDGLPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR SPI2LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR SPI3LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR SPDIFRXPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR USART2LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR USART3LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR UART4LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR UART5LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR I2C1LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR I2C2LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR I2C3LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR FMPI2C1LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR CAN1LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR CAN2LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR CAN3LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR CECLPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR PWRLPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR DACLPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR UART7LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR UART8LPEN LL\_APB1\_GRP1\_DisableClockLowPower
- APB1LPENR RTCAPBLPEN LL\_APB1\_GRP1\_DisableClockLowPower

**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\_TIM1
  - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_USART1
  - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC1
  - LL\_APB2\_GRP1\_PERIPH\_ADC2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC3 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI1
  - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
  - LL\_APB2\_GRP1\_PERIPH\_EXTI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM9
  - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM11
  - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB2ENR TIM1EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR TIM8EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR USART1EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR USART6EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR UART9EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR UART10EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR ADC1EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR ADC2EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR ADC3EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SDIOEN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SPI1EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SPI4EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SYSCFGEN LL\_APB2\_GRP1\_EnableClock
- APB2ENR EXTITEN LL\_APB2\_GRP1\_EnableClock
- APB2ENR TIM9EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR TIM10EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR TIM11EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SPI5EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SPI6EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SAI1EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR SAI2EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR LTDCEN LL\_APB2\_GRP1\_EnableClock
- APB2ENR DSIEN LL\_APB2\_GRP1\_EnableClock
- APB2ENR DFSDM1EN LL\_APB2\_GRP1\_EnableClock
- APB2ENR DFSDM2EN 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\_TIM1
  - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_USART1
  - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC1
  - LL\_APB2\_GRP1\_PERIPH\_ADC2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC3 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI1
  - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
  - LL\_APB2\_GRP1\_PERIPH\_EXTI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM9
  - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM11
  - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)

(\*) value not defined in all devices.

## Return values

- **State:** of Periphs (1 or 0).

**Reference Manual to LL API cross reference:**

- APB2ENR TIM1EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR TIM8EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR USART1EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR USART6EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR UART9EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR UART10EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR ADC1EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR ADC2EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR ADC3EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SDIOEN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SPI1EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SPI4EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SYSCFGEN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR EXTITEN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR TIM9EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR TIM10EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR TIM11EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SPI5EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SPI6EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SAI1EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR SAI2EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR LTDCEN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR DSIEN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR DFSDM1EN LL\_APB2\_GRP1\_IsEnabledClock
- APB2ENR DFSDM2EN 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\_TIM1
  - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_USART1
  - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC1
  - LL\_APB2\_GRP1\_PERIPH\_ADC2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC3 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI1
  - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
  - LL\_APB2\_GRP1\_PERIPH\_EXTI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM9
  - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM11
  - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB2ENR TIM1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR TIM8EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR USART1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR USART6EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR UART9EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR UART10EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR ADC1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR ADC2EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR ADC3EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SDIOEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SPI1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SPI4EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SYSCFGEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR EXTITEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR TIM9EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR TIM10EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR TIM11EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SPI5EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SPI6EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SAI1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR SAI2EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR LTDCEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR DSIEN LL\_APB2\_GRP1\_DisableClock
- APB2ENR DFSDM1EN LL\_APB2\_GRP1\_DisableClock
- APB2ENR DFSDM2EN 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\_ALL
    - LL\_APB2\_GRP1\_PERIPH\_TIM1
    - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_USART1
    - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_ADC
    - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SPI1
    - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
    - LL\_APB2\_GRP1\_PERIPH\_TIM9
    - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_TIM11
    - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
    - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
    - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)
- (\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB2RSTR TIM1RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR TIM8RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR USART1RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR USART6RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR UART9RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR UART10RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR ADCRST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SDIORST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SPI1RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SPI4RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SYSCFGRST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR TIM9RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR TIM10RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR TIM11RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SPI5RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SPI6RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SAI1RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR SAI2RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR LTDCRST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR DSIRST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR DFSDM1RST LL\_APB2\_GRP1\_ForceReset
- APB2RSTR DFSDM2RST 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\_ALL
    - LL\_APB2\_GRP1\_PERIPH\_TIM1
    - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_USART1
    - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_ADC
    - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SPI1
    - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
    - LL\_APB2\_GRP1\_PERIPH\_EXTI (\*)
    - LL\_APB2\_GRP1\_PERIPH\_TIM9
    - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_TIM11
    - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
    - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
    - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
    - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)
- (\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB2RSTR TIM1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR TIM8RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR USART1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR USART6RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR UART9RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR UART10RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR ADCRST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SDIORST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SPI1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SPI4RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SYSCFGRST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR TIM9RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR TIM10RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR TIM11RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SPI5RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SPI6RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SAI1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR SAI2RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR LTDCRST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR DSIRST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR DFSDM1RST LL\_APB2\_GRP1\_ReleaseReset
- APB2RSTR DFSDM2RST LL\_APB2\_GRP1\_ReleaseReset

**LL\_APB2\_GRP1\_EnableClockLowPower**
**Function name**

```
__STATIC_INLINE void LL_APB2_GRP1_EnableClockLowPower (uint32_t Periphs)
```

**Function description**

Enable APB2 peripheral clocks in low-power mode.

## Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL\_APB2\_GRP1\_PERIPH\_TIM1
  - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_USART1
  - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC1
  - LL\_APB2\_GRP1\_PERIPH\_ADC2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC3 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI1
  - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
  - LL\_APB2\_GRP1\_PERIPH\_EXTI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM9
  - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM11
  - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB2LPENR TIM1LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR TIM8LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR USART1LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR USART6LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR UART9LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR UART10LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR ADC1LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR ADC2LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR ADC3LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SDIOLPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SPI1LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SPI4LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SYSCFGLPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR EXTITLPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR TIM9LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR TIM10LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR TIM11LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SPI5LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SPI6LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SAI1LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR SAI2LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR LTDCLPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR DSILPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR DFSDM1LPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR DSILPEN LL\_APB2\_GRP1\_EnableClockLowPower
- APB2LPENR DFSDM2LPEN LL\_APB2\_GRP1\_EnableClockLowPower

**LL\_APB2\_GRP1\_DisableClockLowPower**
**Function name**

```
__STATIC_INLINE void LL_APB2_GRP1_DisableClockLowPower (uint32_t Periphs)
```

**Function description**

Disable APB2 peripheral clocks in low-power mode.

## Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL\_APB2\_GRP1\_PERIPH\_TIM1
  - LL\_APB2\_GRP1\_PERIPH\_TIM8 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_USART1
  - LL\_APB2\_GRP1\_PERIPH\_USART6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART9 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_UART10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC1
  - LL\_APB2\_GRP1\_PERIPH\_ADC2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_ADC3 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SDIO (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI1
  - LL\_APB2\_GRP1\_PERIPH\_SPI4 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SYSCFG
  - LL\_APB2\_GRP1\_PERIPH\_EXTI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM9
  - LL\_APB2\_GRP1\_PERIPH\_TIM10 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_TIM11
  - LL\_APB2\_GRP1\_PERIPH\_SPI5 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SPI6 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_SAI2 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_LTDC (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DSI (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM1 (\*)
  - LL\_APB2\_GRP1\_PERIPH\_DFSDM2 (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

**Reference Manual to LL API cross reference:**

- APB2LPENR TIM1LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR TIM8LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR USART1LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR USART6LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR UART9LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR UART10LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR ADC1LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR ADC2LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR ADC3LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SDIOLPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SPI1LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SPI4LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SYSCFGLPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR EXTITLPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR TIM9LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR TIM10LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR TIM11LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SPI5LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SPI6LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SAI1LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR SAI2LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR LTDCLPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR DSILPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR DFSDM1LPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR DSILPEN LL\_APB2\_GRP1\_DisableClockLowPower
- APB2LPENR DFSDM2LPEN LL\_APB2\_GRP1\_DisableClockLowPower

## 74.2 BUS Firmware driver defines

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

### 74.2.1 BUS

BUS

***AHB1 GRP1 PERIPH***

***LL\_AHB1\_GRP1\_PERIPH\_ALL***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOA***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOB***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOC***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOD***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOE***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOF***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOG***

***LL\_AHB1\_GRP1\_PERIPH\_GPIOH***

LL\_AHB1\_GRP1\_PERIPH\_GPIOI  
LL\_AHB1\_GRP1\_PERIPH\_GPIOJ  
LL\_AHB1\_GRP1\_PERIPH\_GPIOK  
LL\_AHB1\_GRP1\_PERIPH\_CRC  
LL\_AHB1\_GRP1\_PERIPH\_BKPSRAM  
LL\_AHB1\_GRP1\_PERIPH\_CCMDATARAM  
LL\_AHB1\_GRP1\_PERIPH\_DMA1  
LL\_AHB1\_GRP1\_PERIPH\_DMA2  
LL\_AHB1\_GRP1\_PERIPH\_DMA2D  
LL\_AHB1\_GRP1\_PERIPH\_ETHMAC  
LL\_AHB1\_GRP1\_PERIPH\_ETHMACTX  
LL\_AHB1\_GRP1\_PERIPH\_ETHMACRX  
LL\_AHB1\_GRP1\_PERIPH\_ETHMACPTP  
LL\_AHB1\_GRP1\_PERIPH\_OTGHS  
LL\_AHB1\_GRP1\_PERIPH\_OTGHSULPI  
LL\_AHB1\_GRP1\_PERIPH\_FLITF  
LL\_AHB1\_GRP1\_PERIPH\_SRAM1  
LL\_AHB1\_GRP1\_PERIPH\_SRAM2  
LL\_AHB1\_GRP1\_PERIPH\_SRAM3

***AHB2 GRP1 PERIPH***

LL\_AHB2\_GRP1\_PERIPH\_ALL  
LL\_AHB2\_GRP1\_PERIPH\_DCMI  
LL\_AHB2\_GRP1\_PERIPH\_Cryp  
LL\_AHB2\_GRP1\_PERIPH\_HASH  
LL\_AHB2\_GRP1\_PERIPH\_RNG  
LL\_AHB2\_GRP1\_PERIPH\_OTGFS

***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\_TIM12

LL\_APB1\_GRP1\_PERIPH\_TIM13

LL\_APB1\_GRP1\_PERIPH\_TIM14

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\_I2C3

LL\_APB1\_GRP1\_PERIPH\_CAN1

LL\_APB1\_GRP1\_PERIPH\_CAN2

LL\_APB1\_GRP1\_PERIPH\_PWR

LL\_APB1\_GRP1\_PERIPH\_DAC1

LL\_APB1\_GRP1\_PERIPH\_UART7

LL\_APB1\_GRP1\_PERIPH\_UART8

***APB2 GRP1 PERIPH***



LL\_APB2\_GRP1\_PERIPH\_ALL  
LL\_APB2\_GRP1\_PERIPH\_TIM1  
LL\_APB2\_GRP1\_PERIPH\_TIM8  
LL\_APB2\_GRP1\_PERIPH\_USART1  
LL\_APB2\_GRP1\_PERIPH\_USART6  
LL\_APB2\_GRP1\_PERIPH\_ADC1  
LL\_APB2\_GRP1\_PERIPH\_ADC2  
LL\_APB2\_GRP1\_PERIPH\_ADC3  
LL\_APB2\_GRP1\_PERIPH\_SDIO  
LL\_APB2\_GRP1\_PERIPH\_SPI1  
LL\_APB2\_GRP1\_PERIPH\_SPI4  
LL\_APB2\_GRP1\_PERIPH\_SYSCFG  
LL\_APB2\_GRP1\_PERIPH\_TIM9  
LL\_APB2\_GRP1\_PERIPH\_TIM10  
LL\_APB2\_GRP1\_PERIPH\_TIM11  
LL\_APB2\_GRP1\_PERIPH\_SPI5  
LL\_APB2\_GRP1\_PERIPH\_SPI6  
LL\_APB2\_GRP1\_PERIPH\_SAI1  
LL\_APB2\_GRP1\_PERIPH\_LTDC  
LL\_APB2\_GRP1\_PERIPH\_DSI  
LL\_APB2\_GRP1\_PERIPH\_ADC

## 75 LL CORTEX Generic Driver

### 75.1 CORTEX Firmware driver API description

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

#### 75.1.1 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 rmpn 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\_GetConstant

#### Function name

`__STATIC_INLINE uint32_t LL_CPUID_GetConstant (void )`

#### Function description

Get Constant 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\_GetConstant

### 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

## 75.2 CORTEX Firmware driver defines

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

### 75.2.1 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\_HARDFFAULT\_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 &amp; user (Full access)

**LL\_MPU\_REGION\_PRIV\_RO**

RO privileged (privileged read only)

**LL\_MPU\_REGION\_PRIV\_RO\_URO**

RO privileged &amp; 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

## 76 LL CRC Generic Driver

### 76.1 CRC Firmware driver API description

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

#### 76.1.1 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\_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\_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\_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 8-bit data register).

**Notes**

- This register can be used as a temporary storage location for one byte.

**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 (8-bit) between Min\_Data=0 and Max\_Data=0xFF

**Return values**

- **None:**

**Notes**

- This register can be used as a temporary storage location for one byte.

**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

## 76.2 CRC Firmware driver defines

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

### 76.2.1 CRC

CRC

***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



## 77 LL DAC Generic Driver

### 77.1 DAC Firmware driver registers structures

#### 77.1.1 LL\_DAC\_InitTypeDef

*LL\_DAC\_InitTypeDef* is defined in the `stm32f4xx_ll_dac.h`

##### Data Fields

- *uint32\_t TriggerSource*
- *uint32\_t WaveAutoGeneration*
- *uint32\_t WaveAutoGenerationConfig*
- *uint32\_t OutputBuffer*

##### Field Documentation

- *uint32\_t LL\_DAC\_InitTypeDef::TriggerSource*  
Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external IP (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()`.
- *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](#)  
**Note:**  
– If waveform automatic generation mode is disabled, this parameter is discarded.  
This feature can be modified afterwards using unitary function `LL_DAC_SetWaveNoiseLFSR()` or `LL_DAC_SetWaveTriangleAmplitude()`, 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()`.

### 77.2 DAC Firmware driver API description

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

#### 77.2.1 Detailed description of functions

##### 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 devices. 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\_TIM8\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM7\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM6\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM5\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM4\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM2\_TRGO
  - LL\_DAC\_TRIG\_EXT\_EXTI\_LINE9

### 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 devices. 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\_TIM8\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM7\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM6\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM5\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM4\_TRGO
  - LL\_DAC\_TRIG\_EXT\_TIM2\_TRGO
  - LL\_DAC\_TRIG\_EXT\_EXTI\_LINE9

### 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 devices. 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

#### 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 devices. 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

### 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 devices. 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

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 devices. 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 devices. 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

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 devices. 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\_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 devices. 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:**

### Reference Manual to LL API cross reference:

- CR BOFF1 LL\_DAC\_SetOutputBuffer
- CR BOFF2 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 devices. 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 BOFF1 LL\_DAC\_GetOutputBuffer
- CR BOFF2 LL\_DAC\_GetOutputBuffer

#### 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 devices. 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 devices. 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 devices. 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\_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 devices. 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 devices. 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 devices. 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 devices. 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\_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 devices. 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 or 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 devices. 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 devices. 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 devices. Refer to device datasheet for channels availability.

### Return values

- **None:**

### Notes

- Preliminarily, DAC trigger must be set to software trigger using function LL\_DAC\_SetTriggerSource() 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\_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 devices. Refer to device datasheet for channels availability.
- **Data:** Value between Min\_Data=0x000 and Max\_Data=0xFFFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- DHR12R1 DAC1DHR LL\_DAC\_ConvertData12RightAligned
- DHR12R2 DAC2DHR 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 devices. Refer to device datasheet for channels availability.
- **Data:** Value between Min\_Data=0x000 and Max\_Data=0xFFFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- DHR12L1 DAC1DHR LL\_DAC\_ConvertData12LeftAligned
- DHR12L2 DAC2DHR 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 devices. 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 devices. 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\_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 instance.

### 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 devices. 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

- The setting of these parameters by function LL\_DAC\_Init() is conditioned to DAC state: DAC instance 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:**

## 77.3 DAC Firmware driver defines

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

**77.3.1 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\_DMAUDR2**

DAC channel 2 flag DMA underrun

**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 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 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 trigger source**

#### LL\_DAC\_TRIG\_SOFTWARE

DAC channel conversion trigger internal (SW start)

#### LL\_DAC\_TRIG\_EXT\_TIM2\_TRGO

DAC channel conversion trigger from external IP: TIM2 TRGO.

#### LL\_DAC\_TRIG\_EXT\_TIM8\_TRGO

DAC channel conversion trigger from external IP: TIM8 TRGO.

#### LL\_DAC\_TRIG\_EXT\_TIM4\_TRGO

DAC channel conversion trigger from external IP: TIM4 TRGO.

#### LL\_DAC\_TRIG\_EXT\_TIM6\_TRGO

DAC channel conversion trigger from external IP: TIM6 TRGO.

#### LL\_DAC\_TRIG\_EXT\_TIM7\_TRGO

DAC channel conversion trigger from external IP: TIM7 TRGO.

#### LL\_DAC\_TRIG\_EXT\_TIM5\_TRGO

DAC channel conversion trigger from external IP: TIM5 TRGO.

#### LL\_DAC\_TRIG\_EXT\_EXTI\_LINE9

DAC channel conversion trigger from external IP: external interrupt line 9.

#### **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.

#### **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: (value "2" depending on DAC channel 2 availability)

**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 (value "2" depending on DAC channel 2 availability)

**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()`.

### **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

## 78 LL DMA2D Generic Driver

### 78.1 DMA2D Firmware driver registers structures

#### 78.1.1 LL\_DMA2D\_InitTypeDef

*LL\_DMA2D\_InitTypeDef* is defined in the `stm32f4xx_ll_dma2d.h`

##### Data Fields

- *uint32\_t Mode*
- *uint32\_t ColorMode*
- *uint32\_t OutputBlue*
- *uint32\_t OutputGreen*
- *uint32\_t OutputRed*
- *uint32\_t OutputAlpha*
- *uint32\_t OutputMemoryAddress*
- *uint32\_t LineOffset*
- *uint32\_t NbrOfLines*
- *uint32\_t NbrOfPixelsPerLines*

##### Field Documentation

- *uint32\_t LL\_DMA2D\_InitTypeDef::Mode*  
Specifies the DMA2D transfer mode.
  - This parameter can be one value of [DMA2D\\_LL\\_EC\\_MODE](#).
 This parameter can be modified afterwards using unitary function `LL_DMA2D_SetMode()`.
- *uint32\_t LL\_DMA2D\_InitTypeDef::ColorMode*  
Specifies the color format of the output image.
  - This parameter can be one value of [DMA2D\\_LL\\_EC\\_OUTPUT\\_COLOR\\_MODE](#).
 This parameter can be modified afterwards using unitary function `LL_DMA2D_SetOutputColorMode()`.
- *uint32\_t LL\_DMA2D\_InitTypeDef::OutputBlue*  
Specifies the Blue value of the output image.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if `ARGB8888` color mode is selected.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if `RGB888` color mode is selected.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if `RGB565` color mode is selected.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if `ARGB1555` color mode is selected.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x0F` if `ARGB4444` color mode is selected.

This parameter can be modified afterwards using unitary function `LL_DMA2D_SetOutputColor()` or configuration function `LL_DMA2D_ConfigOutputColor()`.

- ***uint32\_t LL\_DMA2D\_InitTypeDef::OutputGreen***  
 Specifies the Green value of the output image.

  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x3F if RGB565 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x0F if ARGB4444 color mode is selected.

This parameter can be modified afterwards using unitary function **LL\_DMA2D\_SetOutputColor()** or configuration function **LL\_DMA2D\_ConfigOutputColor()**.
- ***uint32\_t LL\_DMA2D\_InitTypeDef::OutputRed***  
 Specifies the Red value of the output image.

  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x0F if ARGB4444 color mode is selected.

This parameter can be modified afterwards using unitary function **LL\_DMA2D\_SetOutputColor()** or configuration function **LL\_DMA2D\_ConfigOutputColor()**.
- ***uint32\_t LL\_DMA2D\_InitTypeDef::OutputAlpha***  
 Specifies the Alpha channel of the output image.

  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x01 if ARGB1555 color mode is selected.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x0F if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

This parameter can be modified afterwards using unitary function **LL\_DMA2D\_SetOutputColor()** or configuration function **LL\_DMA2D\_ConfigOutputColor()**.
- ***uint32\_t LL\_DMA2D\_InitTypeDef::OutputMemoryAddress***  
 Specifies the memory address.

  - This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFFFFFF.

This parameter can be modified afterwards using unitary function **LL\_DMA2D\_SetOutputMemAddr()**.
- ***uint32\_t LL\_DMA2D\_InitTypeDef::LineOffset***  
 Specifies the output line offset value.

  - This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x3FFF.

This parameter can be modified afterwards using unitary function **LL\_DMA2D\_SetLineOffset()**.
- ***uint32\_t LL\_DMA2D\_InitTypeDef::NbrOfLines***  
 Specifies the number of lines of the area to be transferred.

  - This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.

This parameter can be modified afterwards using unitary function **LL\_DMA2D\_SetNbrOfLines()**.

- ***uint32\_t LL\_DMA2D\_InitTypeDef::NbrOfPixelsPerLines***  
Specifies the number of pixels per lines of the area to be transferred.
  - This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0x3FFF`.
 This parameter can be modified afterwards using unitary function `LL_DMA2D_SetNbrOfPixelsPerLines()`.

### 78.1.2 LL\_DMA2D\_LayerCfgTypeDef

*LL\_DMA2D\_LayerCfgTypeDef* is defined in the `stm32f4xx_ll_dma2d.h`

#### Data Fields

- ***uint32\_t MemoryAddress***
- ***uint32\_t LineOffset***
- ***uint32\_t ColorMode***
- ***uint32\_t CLUTColorMode***
- ***uint32\_t CLUTSize***
- ***uint32\_t AlphaMode***
- ***uint32\_t Alpha***
- ***uint32\_t Blue***
- ***uint32\_t Green***
- ***uint32\_t Red***
- ***uint32\_t CLUTMemoryAddress***

#### Field Documentation

- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::MemoryAddress***  
Specifies the foreground or background memory address.
  - This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFFFFFF`.
 This parameter can be modified afterwards using unitary functions
  - `LL_DMA2D_FGND_SetMemAddr()` for foreground layer,
  - `LL_DMA2D_BGND_SetMemAddr()` for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::LineOffset***  
Specifies the foreground or background line offset value.
  - This parameter must be a number between `Min_Data = 0x0000` and `Max_Data = 0x3FFF`.
 This parameter can be modified afterwards using unitary functions
  - `LL_DMA2D_FGND_SetLineOffset()` for foreground layer,
  - `LL_DMA2D_BGND_SetLineOffset()` for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::ColorMode***  
Specifies the foreground or background color mode.
  - This parameter can be one value of `DMA2D_LL_EC_INPUT_COLOR_MODE`.
 This parameter can be modified afterwards using unitary functions
  - `LL_DMA2D_FGND_SetColorMode()` for foreground layer,
  - `LL_DMA2D_BGND_SetColorMode()` for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::CLUTColorMode***  
Specifies the foreground or background CLUT color mode.
  - This parameter can be one value of `DMA2D_LL_EC_CLUT_COLOR_MODE`.
 This parameter can be modified afterwards using unitary functions
  - `LL_DMA2D_FGND_SetCLUTColorMode()` for foreground layer,
  - `LL_DMA2D_BGND_SetCLUTColorMode()` for background layer.

- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::CLUTSize***  
 Specifies the foreground or background CLUT size.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetCLUTSize()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetCLUTSize()** for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::AlphaMode***  
 Specifies the foreground or background alpha mode.
  - This parameter can be one value of **DMA2D\_LL\_EC\_ALPHA\_MODE**.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetAlphaMode()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetAlphaMode()** for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::Alpha***  
 Specifies the foreground or background Alpha value.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetAlpha()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetAlpha()** for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::Blue***  
 Specifies the foreground or background Blue color value.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetBlueColor()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetBlueColor()** for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::Green***  
 Specifies the foreground or background Green color value.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetGreenColor()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetGreenColor()** for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::Red***  
 Specifies the foreground or background Red color value.
  - This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetRedColor()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetRedColor()** for background layer.
- ***uint32\_t LL\_DMA2D\_LayerCfgTypeDef::CLUTMemoryAddress***  
 Specifies the foreground or background CLUT memory address.
  - This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFFFFFF.
 This parameter can be modified afterwards using unitary functions
  - **LL\_DMA2D\_FGND\_SetCLUTMemAddr()** for foreground layer,
  - **LL\_DMA2D\_BGND\_SetCLUTMemAddr()** for background layer.

### 78.1.3 LL\_DMA2D\_ColorTypeDef

**LL\_DMA2D\_ColorTypeDef** is defined in the stm32f4xx\_ll\_dma2d.h

#### Data Fields

- ***uint32\_t ColorMode***
- ***uint32\_t OutputBlue***

- ***uint32\_t OutputGreen***
- ***uint32\_t OutputRed***
- ***uint32\_t OutputAlpha***

#### Field Documentation

- ***uint32\_t LL\_DMA2D\_ColorTypeDef::ColorMode***

Specifies the color format of the output image.

- This parameter can be one value of ***DMA2D\_LL\_EC\_OUTPUT\_COLOR\_MODE***.

This parameter can be modified afterwards using unitary function ***LL\_DMA2D\_SetOutputColorMode()***.

- ***uint32\_t LL\_DMA2D\_ColorTypeDef::OutputBlue***

Specifies the Blue value of the output image.

- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if ARGB8888 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if RGB888 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if RGB565 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if ARGB1555 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x0F` if ARGB4444 color mode is selected.

This parameter can be modified afterwards using unitary function ***LL\_DMA2D\_SetOutputColor()*** or configuration function ***LL\_DMA2D\_ConfigOutputColor()***.

- ***uint32\_t LL\_DMA2D\_ColorTypeDef::OutputGreen***

Specifies the Green value of the output image.

- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if ARGB8888 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if RGB888 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x3F` if RGB565 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if ARGB1555 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x0F` if ARGB4444 color mode is selected.

This parameter can be modified afterwards using unitary function ***LL\_DMA2D\_SetOutputColor()*** or configuration function ***LL\_DMA2D\_ConfigOutputColor()***.

- ***uint32\_t LL\_DMA2D\_ColorTypeDef::OutputRed***

Specifies the Red value of the output image.

- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if ARGB8888 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if RGB888 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if RGB565 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x1F` if ARGB1555 color mode is selected.
- This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x0F` if ARGB4444 color mode is selected.

This parameter can be modified afterwards using unitary function ***LL\_DMA2D\_SetOutputColor()*** or configuration function ***LL\_DMA2D\_ConfigOutputColor()***.

- **`uint32_t LL_DMA2D_ColorTypeDef::OutputAlpha`**  
Specifies the Alpha channel of the output image.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0xFF` if ARGB8888 color mode is selected.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x01` if ARGB1555 color mode is selected.
  - This parameter must be a number between `Min_Data = 0x00` and `Max_Data = 0x0F` if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

This parameter can be modified afterwards using unitary function `LL_DMA2D_SetOutputColor()` or configuration function `LL_DMA2D_ConfigOutputColor()`.

## 78.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

### 78.2.1 Detailed description of functions

#### `LL_DMA2D_Start`

##### Function name

```
__STATIC_INLINE void LL_DMA2D_Start (DMA2D_TypeDef * DMA2Dx)
```

##### Function description

Start a DMA2D transfer.

##### Parameters

- **DMA2Dx:** DMA2D Instance

##### Return values

- **None:**

##### Reference Manual to LL API cross reference:

- CR START `LL_DMA2D_Start`

#### `LL_DMA2D_IsTransferOngoing`

##### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsTransferOngoing (DMA2D_TypeDef * DMA2Dx)
```

##### Function description

Indicate if a DMA2D transfer is ongoing.

##### Parameters

- **DMA2Dx:** DMA2D Instance

##### Return values

- **State:** of bit (1 or 0).

##### Reference Manual to LL API cross reference:

- CR START `LL_DMA2D_IsTransferOngoing`

#### `LL_DMA2D_Suspend`

##### Function name

```
__STATIC_INLINE void LL_DMA2D_Suspend (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Suspend DMA2D transfer.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Notes

- This API can be used to suspend automatic foreground or background CLUT loading.

### Reference Manual to LL API cross reference:

- CR SUSP LL\_DMA2D\_Suspend

### LL\_DMA2D\_Resume

### Function name

```
__STATIC_INLINE void LL_DMA2D_Resume (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Resume DMA2D transfer.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Notes

- This API can be used to resume automatic foreground or background CLUT loading.

### Reference Manual to LL API cross reference:

- CR SUSP LL\_DMA2D\_Resume

### LL\_DMA2D\_IsSuspended

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsSuspended (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Indicate if DMA2D transfer is suspended.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is suspended.

### Reference Manual to LL API cross reference:

- CR SUSP LL\_DMA2D\_IsSuspended



## LL\_DMA2D\_Abort

### Function name

```
__STATIC_INLINE void LL_DMA2D_Abort (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Abort DMA2D transfer.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Notes

- This API can be used to abort automatic foreground or background CLUT loading.

### Reference Manual to LL API cross reference:

- CR ABORT LL\_DMA2D\_Abort

## LL\_DMA2D\_IsAborted

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsAborted (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Indicate if DMA2D transfer is aborted.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is aborted.

### Reference Manual to LL API cross reference:

- CR ABORT LL\_DMA2D\_IsAborted

## LL\_DMA2D\_SetMode

### Function name

```
__STATIC_INLINE void LL_DMA2D_SetMode (DMA2D_TypeDef * DMA2Dx, uint32_t Mode)
```

### Function description

Set DMA2D mode.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Mode:** This parameter can be one of the following values:
  - LL\_DMA2D\_MODE\_M2M
  - LL\_DMA2D\_MODE\_M2M\_PFC
  - LL\_DMA2D\_MODE\_M2M\_BLEND
  - LL\_DMA2D\_MODE\_R2M

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR MODE LL\_DMA2D\_SetMode

### LL\_DMA2D\_GetMode

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_GetMode (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D mode.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_MODE\_M2M
  - LL\_DMA2D\_MODE\_M2M\_PFC
  - LL\_DMA2D\_MODE\_M2M\_BLEND
  - LL\_DMA2D\_MODE\_R2M

### Reference Manual to LL API cross reference:

- CR MODE LL\_DMA2D\_GetMode

### LL\_DMA2D\_SetOutputColorMode

### Function name

```
__STATIC_INLINE void LL_DMA2D_SetOutputColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)
```

### Function description

Set DMA2D output color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB565
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- OPFCCR CM LL\_DMA2D\_SetOutputColorMode

### LL\_DMA2D\_GetOutputColorMode

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColorMode (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D output color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB565
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444

### Reference Manual to LL API cross reference:

- OPFCCR CM LL\_DMA2D\_GetOutputColorMode

### LL\_DMA2D\_SetLineOffset

### Function name

```
__STATIC_INLINE void LL_DMA2D_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)
```

### Function description

Set DMA2D line offset, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min\_Data=0 and Max\_Data=0x3FFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- OOR LO LL\_DMA2D\_SetLineOffset

### LL\_DMA2D\_GetLineOffset

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffset (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D line offset, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Line:** offset value between Min\_Data=0 and Max\_Data=0x3FFF

### Reference Manual to LL API cross reference:

- OOR LO LL\_DMA2D\_GetLineOffset

### LL\_DMA2D\_SetNbrOfPixelsPerLines

### Function name

```
__STATIC_INLINE void LL_DMA2D_SetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfPixelsPerLines)
```

### Function description

Set DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfPixelsPerLines:** Value between Min\_Data=0 and Max\_Data=0x3FFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- NLR PL LL\_DMA2D\_SetNbrOfPixelsPerLines

**LL\_DMA2D\_GetNbrOfPixelsPerLines**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_GetNbrOfPixelsPerLines (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Return DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits)

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Number:** of pixels per lines value between Min\_Data=0 and Max\_Data=0x3FFF

### Reference Manual to LL API cross reference:

- NLR PL LL\_DMA2D\_GetNbrOfPixelsPerLines

**LL\_DMA2D\_SetNbrOfLines**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA2D\_SetNbrOfLines (DMA2D\_TypeDef \* DMA2Dx, uint32\_t NbrOfLines)**

### Function description

Set DMA2D number of lines, expressed on 16 bits ([15:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min\_Data=0 and Max\_Data=0xFFFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- NLR NL LL\_DMA2D\_SetNbrOfLines

**LL\_DMA2D\_GetNbrOfLines**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_GetNbrOfLines (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Return DMA2D number of lines, expressed on 16 bits ([15:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Number:** of lines value between Min\_Data=0 and Max\_Data=0xFFFF

**Reference Manual to LL API cross reference:**

- NLR NL LL\_DMA2D\_GetNbrOfLines

**LL\_DMA2D\_SetOutputMemAddr**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_SetOutputMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t OutputMemoryAddress)
```

**Function description**

Set DMA2D output memory address, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **OutputMemoryAddress:** Value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- OMAR MA LL\_DMA2D\_SetOutputMemAddr

**LL\_DMA2D\_GetOutputMemAddr**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_GetOutputMemAddr (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Get DMA2D output memory address, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Output:** memory address value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Reference Manual to LL API cross reference:**

- OMAR MA LL\_DMA2D\_GetOutputMemAddr

**LL\_DMA2D\_SetOutputColor**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_SetOutputColor (DMA2D_TypeDef * DMA2Dx, uint32_t OutputColor)
```

**Function description**

Set DMA2D output color, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **OutputColor:** Value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Return values**

- **None:**

**Notes**

- Output color format depends on output color mode, ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444.
- LL\_DMA2D\_ConfigOutputColor() API may be used instead if colors values formatting with respect to color mode is not done by the user code.

**Reference Manual to LL API cross reference:**

- OCOLR BLUE LL\_DMA2D\_SetOutputColor
- OCOLR GREEN LL\_DMA2D\_SetOutputColor
- OCOLR RED LL\_DMA2D\_SetOutputColor
- OCOLR ALPHA LL\_DMA2D\_SetOutputColor

**LL\_DMA2D\_GetOutputColor**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColor (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Get DMA2D output color, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Output:** color value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Notes**

- Alpha channel and red, green, blue color values must be retrieved from the returned value based on the output color mode (ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444) as set by LL\_DMA2D\_SetOutputColorMode.

**Reference Manual to LL API cross reference:**

- OCOLR BLUE LL\_DMA2D\_GetOutputColor
- OCOLR GREEN LL\_DMA2D\_GetOutputColor
- OCOLR RED LL\_DMA2D\_GetOutputColor
- OCOLR ALPHA LL\_DMA2D\_GetOutputColor

**LL\_DMA2D\_SetLineWatermark**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_SetLineWatermark (DMA2D_TypeDef * DMA2Dx, uint32_t LineWatermark)
```

**Function description**

Set DMA2D line watermark, expressed on 16 bits ([15:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **LineWatermark:** Value between Min\_Data=0 and Max\_Data=0xFFFF

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LWR LW LL\_DMA2D\_SetLineWatermark

### LL\_DMA2D\_GetLineWatermark

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_GetLineWatermark (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Return DMA2D line watermark, expressed on 16 bits ([15:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Line:** watermark value between Min\_Data=0 and Max\_Data=0xFFFF

#### Reference Manual to LL API cross reference:

- LWR LW LL\_DMA2D\_GetLineWatermark

### LL\_DMA2D\_SetDeadTime

#### Function name

```
__STATIC_INLINE void LL_DMA2D_SetDeadTime (DMA2D_TypeDef * DMA2Dx, uint32_t DeadTime)
```

#### Function description

Set DMA2D dead time, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **DeadTime:** Value between Min\_Data=0 and Max\_Data=0xFF

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- AMTCR DT LL\_DMA2D\_SetDeadTime

### LL\_DMA2D\_GetDeadTime

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_GetDeadTime (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Return DMA2D dead time, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Dead:** time value between Min\_Data=0 and Max\_Data=0xFF

#### Reference Manual to LL API cross reference:

- AMTCR DT LL\_DMA2D\_GetDeadTime

### LL\_DMA2D\_EnableDeadTime

#### Function name

```
__STATIC_INLINE void LL_DMA2D_EnableDeadTime (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Enable DMA2D dead time functionality.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AMTCR EN LL\_DMA2D\_EnableDeadTime

### LL\_DMA2D\_DisableDeadTime

### Function name

```
__STATIC_INLINE void LL_DMA2D_DisableDeadTime (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Disable DMA2D dead time functionality.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- AMTCR EN LL\_DMA2D\_DisableDeadTime

### LL\_DMA2D\_IsEnabledDeadTime

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledDeadTime (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Indicate if DMA2D dead time functionality is enabled.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- AMTCR EN LL\_DMA2D\_IsEnabledDeadTime

### LL\_DMA2D\_FGND\_SetMemAddr

### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)
```

### Function description

Set DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min\_Data=0 and Max\_Data=0xFFFFFFFF



#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- FGMMAR MA LL\_DMA2D\_FGND\_SetMemAddr

#### LL\_DMA2D\_FGND\_GetMemAddr

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetMemAddr (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Get DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Foreground:** memory address value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

#### Reference Manual to LL API cross reference:

- FGMMAR MA LL\_DMA2D\_FGND\_GetMemAddr

#### LL\_DMA2D\_FGND\_EnableCLUTLoad

#### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Enable DMA2D foreground CLUT loading.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- FGPFCCR START LL\_DMA2D\_FGND\_EnableCLUTLoad

#### LL\_DMA2D\_FGND\_IsEnabledCLUTLoad

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_IsEnabledCLUTLoad (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Indicate if DMA2D foreground CLUT loading is enabled.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- FGPFCCR START LL\_DMA2D\_FGND\_IsEnabledCLUTLoad

## LL\_DMA2D\_FGND\_SetColorMode

### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)
```

### Function description

Set DMA2D foreground color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_INPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_INPUT\_MODE\_RGB888
  - LL\_DMA2D\_INPUT\_MODE\_RGB565
  - LL\_DMA2D\_INPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_INPUT\_MODE\_ARGB4444
  - LL\_DMA2D\_INPUT\_MODE\_L8
  - LL\_DMA2D\_INPUT\_MODE\_AL44
  - LL\_DMA2D\_INPUT\_MODE\_AL88
  - LL\_DMA2D\_INPUT\_MODE\_L4
  - LL\_DMA2D\_INPUT\_MODE\_A8
  - LL\_DMA2D\_INPUT\_MODE\_A4

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGPFCCR CM LL\_DMA2D\_FGND\_SetColorMode

## LL\_DMA2D\_FGND\_GetColorMode

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetColorMode (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D foreground color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_INPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_INPUT\_MODE\_RGB888
  - LL\_DMA2D\_INPUT\_MODE\_RGB565
  - LL\_DMA2D\_INPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_INPUT\_MODE\_ARGB4444
  - LL\_DMA2D\_INPUT\_MODE\_L8
  - LL\_DMA2D\_INPUT\_MODE\_AL44
  - LL\_DMA2D\_INPUT\_MODE\_AL88
  - LL\_DMA2D\_INPUT\_MODE\_L4
  - LL\_DMA2D\_INPUT\_MODE\_A8
  - LL\_DMA2D\_INPUT\_MODE\_A4

**Reference Manual to LL API cross reference:**

- FGPFCCR CM LL\_DMA2D\_FGND\_GetColorMode

**LL\_DMA2D\_FGND\_SetAlphaMode**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)
```

**Function description**

Set DMA2D foreground alpha mode.

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_ALPHA\_MODE\_NO\_MODIF
  - LL\_DMA2D\_ALPHA\_MODE\_REPLACE
  - LL\_DMA2D\_ALPHA\_MODE\_COMBINE

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- FGPFCCR AM LL\_DMA2D\_FGND\_SetAlphaMode

**LL\_DMA2D\_FGND\_GetAlphaMode**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaMode (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Return DMA2D foreground alpha mode.

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_ALPHA\_MODE\_NO\_MODIF
  - LL\_DMA2D\_ALPHA\_MODE\_REPLACE
  - LL\_DMA2D\_ALPHA\_MODE\_COMBINE

**Reference Manual to LL API cross reference:**

- FGPFCCR AM LL\_DMA2D\_FGND\_GetAlphaMode

**LL\_DMA2D\_FGND\_SetAlpha**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_FGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)
```

**Function description**

Set DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL\_DMA2D\_FGND\_SetAlpha

### LL\_DMA2D\_FGND\_GetAlpha

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlpha (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Alpha:** value between Min\_Data=0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL\_DMA2D\_FGND\_GetAlpha

### LL\_DMA2D\_FGND\_SetLineOffset

### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)
```

### Function description

Set DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min\_Data=0 and Max\_Data=0x3FFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGOR LO LL\_DMA2D\_FGND\_SetLineOffset

### LL\_DMA2D\_FGND\_GetLineOffset

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetLineOffset (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Foreground:** line offset value between Min\_Data=0 and Max\_Data=0x3FFF

### Reference Manual to LL API cross reference:

- FGOR LO LL\_DMA2D\_FGND\_GetLineOffset

## LL\_DMA2D\_FGND\_SetColor

### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)
```

### Function description

Set DMA2D foreground color values, expressed on 24 bits ([23:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min\_Data=0 and Max\_Data=0xFF
- **Green:** Value between Min\_Data=0 and Max\_Data=0xFF
- **Blue:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGCOLR RED LL\_DMA2D\_FGND\_SetColor
- FGCOLR GREEN LL\_DMA2D\_FGND\_SetColor
- FGCOLR BLUE LL\_DMA2D\_FGND\_SetColor

## LL\_DMA2D\_FGND\_SetRedColor

### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)
```

### Function description

Set DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGCOLR RED LL\_DMA2D\_FGND\_SetRedColor

## LL\_DMA2D\_FGND\_GetRedColor

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRedColor (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Red:** color value between Min\_Data=0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- FGCOLR RED LL\_DMA2D\_FGND\_GetRedColor

### LL\_DMA2D\_FGND\_SetGreenColor

#### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)
```

#### Function description

Set DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min\_Data=0 and Max\_Data=0xFF

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- FGCOLOR GREEN LL\_DMA2D\_FGND\_SetGreenColor

### LL\_DMA2D\_FGND\_GetGreenColor

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetGreenColor (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Return DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Green:** color value between Min\_Data=0 and Max\_Data=0xFF

#### Reference Manual to LL API cross reference:

- FGCOLOR GREEN LL\_DMA2D\_FGND\_GetGreenColor

### LL\_DMA2D\_FGND\_SetBlueColor

#### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)
```

#### Function description

Set DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min\_Data=0 and Max\_Data=0xFF

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- FGCOLOR BLUE LL\_DMA2D\_FGND\_SetBlueColor

### LL\_DMA2D\_FGND\_GetBlueColor

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetBlueColor (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Return DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Blue:** color value between Min\_Data=0 and Max\_Data=0xFF

**Reference Manual to LL API cross reference:**

- FGCOLOR BLUE LL\_DMA2D\_FGND\_GetBlueColor

**LL\_DMA2D\_FGND\_SetCLUTMemAddr**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)
```

**Function description**

Set DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- FGCMAR MA LL\_DMA2D\_FGND\_SetCLUTMemAddr

**LL\_DMA2D\_FGND\_GetCLUTMemAddr**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Get DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Foreground:** CLUT memory address value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Reference Manual to LL API cross reference:**

- FGCMAR MA LL\_DMA2D\_FGND\_GetCLUTMemAddr

**LL\_DMA2D\_FGND\_SetCLUTSize**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)
```

**Function description**

Set DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGPFCCR CS LL\_DMA2D\_FGND\_SetCLUTSize

### LL\_DMA2D\_FGND\_GetCLUTSize

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTSize (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Get DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Foreground:** CLUT size value between Min\_Data=0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- FGPFCCR CS LL\_DMA2D\_FGND\_GetCLUTSize

### LL\_DMA2D\_FGND\_SetCLUTColorMode

### Function name

```
__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)
```

### Function description

Set DMA2D foreground CLUT color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_ARGB8888
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_RGB888

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FGPFCCR CCM LL\_DMA2D\_FGND\_SetCLUTColorMode

### LL\_DMA2D\_FGND\_GetCLUTColorMode

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTColorMode (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D foreground CLUT color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance



**Return values**

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_ARGB8888
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_RGB888

**Reference Manual to LL API cross reference:**

- FGPFCCR CCM LL\_DMA2D\_FGND\_GetCLUTColorMode

**LL\_DMA2D\_BGND\_SetMemAddr**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_BGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)
```

**Function description**

Set DMA2D background memory address, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- BGMAR MA LL\_DMA2D\_BGND\_SetMemAddr

**LL\_DMA2D\_BGND\_GetMemAddr**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetMemAddr (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Get DMA2D background memory address, expressed on 32 bits ([31:0] bits).

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **Background:** memory address value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

**Reference Manual to LL API cross reference:**

- BGMAR MA LL\_DMA2D\_BGND\_GetMemAddr

**LL\_DMA2D\_BGND\_EnableCLUTLoad**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_BGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Enable DMA2D background CLUT loading.

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- BGPFFCCR START LL\_DMA2D\_BGND\_EnableCLUTLoad

**LL\_DMA2D\_BGND\_IsEnabledCLUTLoad**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_IsEnabledCLUTLoad (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Indicate if DMA2D background CLUT loading is enabled.

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- BGPFFCCR START LL\_DMA2D\_BGND\_IsEnabledCLUTLoad

**LL\_DMA2D\_BGND\_SetColorMode**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_BGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)
```

**Function description**

Set DMA2D background color mode.

**Parameters**

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_INPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_INPUT\_MODE\_RGB888
  - LL\_DMA2D\_INPUT\_MODE\_RGB565
  - LL\_DMA2D\_INPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_INPUT\_MODE\_ARGB4444
  - LL\_DMA2D\_INPUT\_MODE\_L8
  - LL\_DMA2D\_INPUT\_MODE\_AL44
  - LL\_DMA2D\_INPUT\_MODE\_AL88
  - LL\_DMA2D\_INPUT\_MODE\_L4
  - LL\_DMA2D\_INPUT\_MODE\_A8
  - LL\_DMA2D\_INPUT\_MODE\_A4

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- BGPFFCCR CM LL\_DMA2D\_BGND\_SetColorMode

**LL\_DMA2D\_BGND\_GetColorMode**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetColorMode (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D background color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_INPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_INPUT\_MODE\_RGB888
  - LL\_DMA2D\_INPUT\_MODE\_RGB565
  - LL\_DMA2D\_INPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_INPUT\_MODE\_ARGB4444
  - LL\_DMA2D\_INPUT\_MODE\_L8
  - LL\_DMA2D\_INPUT\_MODE\_AL44
  - LL\_DMA2D\_INPUT\_MODE\_AL88
  - LL\_DMA2D\_INPUT\_MODE\_L4
  - LL\_DMA2D\_INPUT\_MODE\_A8
  - LL\_DMA2D\_INPUT\_MODE\_A4

### Reference Manual to LL API cross reference:

- BGPFCR CM LL\_DMA2D\_BGND\_GetColorMode

### LL\_DMA2D\_BGND\_SetAlphaMode

### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)
```

### Function description

Set DMA2D background alpha mode.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_ALPHA\_MODE\_NO\_MODIF
  - LL\_DMA2D\_ALPHA\_MODE\_REPLACE
  - LL\_DMA2D\_ALPHA\_MODE\_COMBINE

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGPFCR AM LL\_DMA2D\_BGND\_SetAlphaMode

### LL\_DMA2D\_BGND\_GetAlphaMode

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaMode (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D background alpha mode.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_ALPHA\_MODE\_NO\_MODIF
  - LL\_DMA2D\_ALPHA\_MODE\_REPLACE
  - LL\_DMA2D\_ALPHA\_MODE\_COMBINE

### Reference Manual to LL API cross reference:

- BGPFCR AM LL\_DMA2D\_BGND\_GetAlphaMode

### LL\_DMA2D\_BGND\_SetAlpha

### Function name

`__STATIC_INLINE void LL_DMA2D_BGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)`

### Function description

Set DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGPFCR ALPHA LL\_DMA2D\_BGND\_SetAlpha

### LL\_DMA2D\_BGND\_GetAlpha

### Function name

`__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlpha (DMA2D_TypeDef * DMA2Dx)`

### Function description

Return DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Alpha:** value between Min\_Data=0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- BGPFCR ALPHA LL\_DMA2D\_BGND\_GetAlpha

### LL\_DMA2D\_BGND\_SetLineOffset

### Function name

`__STATIC_INLINE void LL_DMA2D_BGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)`

### Function description

Set DMA2D background line offset, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min\_Data=0 and Max\_Data=0x3FFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGOR LO LL\_DMA2D\_BGND\_SetLineOffset

### LL\_DMA2D\_BGND\_GetLineOffset

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetLineOffset (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D background line offset, expressed on 14 bits ([13:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Background:** line offset value between Min\_Data=0 and Max\_Data=0x3FF

### Reference Manual to LL API cross reference:

- BGOR LO LL\_DMA2D\_BGND\_GetLineOffset

### LL\_DMA2D\_BGND\_SetColor

### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)
```

### Function description

Set DMA2D background color values, expressed on 24 bits ([23:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min\_Data=0 and Max\_Data=0xFF
- **Green:** Value between Min\_Data=0 and Max\_Data=0xFF
- **Blue:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGCOLOR RED LL\_DMA2D\_BGND\_SetColor
- BGCOLOR GREEN LL\_DMA2D\_BGND\_SetColor
- BGCOLOR BLUE LL\_DMA2D\_BGND\_SetColor

### LL\_DMA2D\_BGND\_SetRedColor

### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)
```

### Function description

Set DMA2D background red color value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGCOLOR RED LL\_DMA2D\_BGND\_SetRedColor

### LL\_DMA2D\_BGND\_GetRedColor

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRedColor (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D background red color value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Red:** color value between Min\_Data=0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- BGCOLOR RED LL\_DMA2D\_BGND\_GetRedColor

### LL\_DMA2D\_BGND\_SetGreenColor

### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)
```

### Function description

Set DMA2D background green color value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min\_Data=0 and Max\_Data=0xFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGCOLOR GREEN LL\_DMA2D\_BGND\_SetGreenColor

### LL\_DMA2D\_BGND\_GetGreenColor

### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetGreenColor (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Return DMA2D background green color value, expressed on 8 bits ([7:0] bits).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Green:** color value between Min\_Data=0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- BGCOLOR GREEN LL\_DMA2D\_BGND\_GetGreenColor

### LL\_DMA2D\_BGND\_SetBlueColor

#### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)
```

#### Function description

Set DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min\_Data=0 and Max\_Data=0xFF

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- BGCOLOR BLUE LL\_DMA2D\_BGND\_SetBlueColor

### LL\_DMA2D\_BGND\_GetBlueColor

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetBlueColor (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Return DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Blue:** color value between Min\_Data=0 and Max\_Data=0xFF

#### Reference Manual to LL API cross reference:

- BGCOLOR BLUE LL\_DMA2D\_BGND\_GetBlueColor

### LL\_DMA2D\_BGND\_SetCLUTMemAddr

#### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)
```

#### Function description

Set DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- BGCMAR MA LL\_DMA2D\_BGND\_SetCLUTMemAddr

### LL\_DMA2D\_BGND\_GetCLUTMemAddr

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Get DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Background:** CLUT memory address value between Min\_Data=0 and Max\_Data=0xFFFFFFFF

#### Reference Manual to LL API cross reference:

- BGCMAR MA LL\_DMA2D\_BGND\_GetCLUTMemAddr

### LL\_DMA2D\_BGND\_SetCLUTSize

#### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)
```

#### Function description

Set DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min\_Data=0 and Max\_Data=0xFF

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- BGPFFCCR CS LL\_DMA2D\_BGND\_SetCLUTSize

### LL\_DMA2D\_BGND\_GetCLUTSize

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTSize (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Get DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **Background:** CLUT size value between Min\_Data=0 and Max\_Data=0xFF

#### Reference Manual to LL API cross reference:

- BGPFFCCR CS LL\_DMA2D\_BGND\_GetCLUTSize

### LL\_DMA2D\_BGND\_SetCLUTColorMode

#### Function name

```
__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)
```



### Function description

Set DMA2D background CLUT color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_ARGB8888
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_RGB888

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BGPFFCCR CCM LL\_DMA2D\_BGND\_SetCLUTColorMode

**LL\_DMA2D\_BGND\_GetCLUTColorMode**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_BGND\_GetCLUTColorMode (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Return DMA2D background CLUT color mode.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_ARGB8888
  - LL\_DMA2D\_CLUT\_COLOR\_MODE\_RGB888

### Reference Manual to LL API cross reference:

- BGPFFCCR CCM LL\_DMA2D\_BGND\_GetCLUTColorMode

**LL\_DMA2D\_IsActiveFlag\_CE**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsActiveFlag\_CE (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Check if the DMA2D Configuration Error Interrupt Flag is set or not.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- ISR CEIF LL\_DMA2D\_IsActiveFlag\_CE

**LL\_DMA2D\_IsActiveFlag\_CTC**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsActiveFlag\_CTC (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Check if the DMA2D CLUT Transfer Complete Interrupt Flag is set or not.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- ISR CTCIF LL\_DMA2D\_IsActiveFlag\_CTC

**LL\_DMA2D\_IsActiveFlag\_CAE**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsActiveFlag\_CAE (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Check if the DMA2D CLUT Access Error Interrupt Flag is set or not.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- ISR CAEIF LL\_DMA2D\_IsActiveFlag\_CAE

**LL\_DMA2D\_IsActiveFlag\_TW**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsActiveFlag\_TW (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Check if the DMA2D Transfer Watermark Interrupt Flag is set or not.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- ISR TWIF LL\_DMA2D\_IsActiveFlag\_TW

**LL\_DMA2D\_IsActiveFlag\_TC**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsActiveFlag\_TC (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Check if the DMA2D Transfer Complete Interrupt Flag is set or not.

### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- ISR TCIF LL\_DMA2D\_IsActiveFlag\_TC

**LL\_DMA2D\_IsActiveFlag\_TE**

#### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsActiveFlag\_TE (DMA2D\_TypeDef \* DMA2Dx)**

#### Function description

Check if the DMA2D Transfer Error Interrupt Flag is set or not.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- ISR TEIF LL\_DMA2D\_IsActiveFlag\_TE

**LL\_DMA2D\_ClearFlag\_CE**

#### Function name

**\_\_STATIC\_INLINE void LL\_DMA2D\_ClearFlag\_CE (DMA2D\_TypeDef \* DMA2Dx)**

#### Function description

Clear DMA2D Configuration Error Interrupt Flag.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- IFCR CCEIF LL\_DMA2D\_ClearFlag\_CE

**LL\_DMA2D\_ClearFlag\_CTC**

#### Function name

**\_\_STATIC\_INLINE void LL\_DMA2D\_ClearFlag\_CTC (DMA2D\_TypeDef \* DMA2Dx)**

#### Function description

Clear DMA2D CLUT Transfer Complete Interrupt Flag.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- IFCR CCTCIF LL\_DMA2D\_ClearFlag\_CTC

### LL\_DMA2D\_ClearFlag\_CAE

#### Function name

```
__STATIC_INLINE void LL_DMA2D_ClearFlag_CAE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Clear DMA2D CLUT Access Error Interrupt Flag.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- IFCR CAECIF LL\_DMA2D\_ClearFlag\_CAE

### LL\_DMA2D\_ClearFlag\_TW

#### Function name

```
__STATIC_INLINE void LL_DMA2D_ClearFlag_TW (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Clear DMA2D Transfer Watermark Interrupt Flag.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- IFCR CTWIF LL\_DMA2D\_ClearFlag\_TW

### LL\_DMA2D\_ClearFlag\_TC

#### Function name

```
__STATIC_INLINE void LL_DMA2D_ClearFlag_TC (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Clear DMA2D Transfer Complete Interrupt Flag.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- IFCR CTCIF LL\_DMA2D\_ClearFlag\_TC

### LL\_DMA2D\_ClearFlag\_TE

#### Function name

```
__STATIC_INLINE void LL_DMA2D_ClearFlag_TE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Clear DMA2D Transfer Error Interrupt Flag.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- IFCR CTEIF LL\_DMA2D\_ClearFlag\_TE

#### LL\_DMA2D\_EnableIT\_CE

#### Function name

```
__STATIC_INLINE void LL_DMA2D_EnableIT_CE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Enable Configuration Error Interrupt.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR CEIE LL\_DMA2D\_EnableIT\_CE

#### LL\_DMA2D\_EnableIT\_CTC

#### Function name

```
__STATIC_INLINE void LL_DMA2D_EnableIT_CTC (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Enable CLUT Transfer Complete Interrupt.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR CTCIE LL\_DMA2D\_EnableIT\_CTC

#### LL\_DMA2D\_EnableIT\_CAE

#### Function name

```
__STATIC_INLINE void LL_DMA2D_EnableIT_CAE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Enable CLUT Access Error Interrupt.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

**Reference Manual to LL API cross reference:**

- CR CAEIE LL\_DMA2D\_EnableIT\_CAE

**LL\_DMA2D\_EnableIT\_TW**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_EnableIT_TW (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Enable Transfer Watermark Interrupt.

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR TWIE LL\_DMA2D\_EnableIT\_TW

**LL\_DMA2D\_EnableIT\_TC**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_EnableIT_TC (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Enable Transfer Complete Interrupt.

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR TCIE LL\_DMA2D\_EnableIT\_TC

**LL\_DMA2D\_EnableIT\_TE**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_EnableIT_TE (DMA2D_TypeDef * DMA2Dx)
```

**Function description**

Enable Transfer Error Interrupt.

**Parameters**

- **DMA2Dx:** DMA2D Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR TEIE LL\_DMA2D\_EnableIT\_TE

**LL\_DMA2D\_DisableIT\_CE**
**Function name**

```
__STATIC_INLINE void LL_DMA2D_DisableIT_CE (DMA2D_TypeDef * DMA2Dx)
```

### Function description

Disable Configuration Error Interrupt.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR CEIE LL\_DMA2D\_DisableIT\_CE

**LL\_DMA2D\_DisableIT\_CTC**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA2D\_DisableIT\_CTC (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Disable CLUT Transfer Complete Interrupt.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR CTCIE LL\_DMA2D\_DisableIT\_CTC

**LL\_DMA2D\_DisableIT\_CAE**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA2D\_DisableIT\_CAE (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Disable CLUT Access Error Interrupt.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR CAEIE LL\_DMA2D\_DisableIT\_CAE

**LL\_DMA2D\_DisableIT\_TW**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA2D\_DisableIT\_TW (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Disable Transfer Watermark Interrupt.

### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR TWIE LL\_DMA2D\_DisableIT\_TW

#### LL\_DMA2D\_DisableIT\_TC

#### Function name

```
__STATIC_INLINE void LL_DMA2D_DisableIT_TC (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Disable Transfer Complete Interrupt.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR TCIE LL\_DMA2D\_DisableIT\_TC

#### LL\_DMA2D\_DisableIT\_TE

#### Function name

```
__STATIC_INLINE void LL_DMA2D_DisableIT_TE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Disable Transfer Error Interrupt.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR TEIE LL\_DMA2D\_DisableIT\_TE

#### LL\_DMA2D\_IsEnabledIT\_CE

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Check if the DMA2D Configuration Error interrupt source is enabled or disabled.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR CEIE LL\_DMA2D\_IsEnabledIT\_CE



### LL\_DMA2D\_IsEnabledIT\_CTC

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CTC (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Check if the DMA2D CLUT Transfer Complete interrupt source is enabled or disabled.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR CTCIE LL\_DMA2D\_IsEnabledIT\_CTC

### LL\_DMA2D\_IsEnabledIT\_CAE

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CAE (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Check if the DMA2D CLUT Access Error interrupt source is enabled or disabled.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR CAEIE LL\_DMA2D\_IsEnabledIT\_CAE

### LL\_DMA2D\_IsEnabledIT\_TW

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TW (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Check if the DMA2D Transfer Watermark interrupt source is enabled or disabled.

#### Parameters

- **DMA2Dx:** DMA2D Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR TWIE LL\_DMA2D\_IsEnabledIT\_TW

### LL\_DMA2D\_IsEnabledIT\_TC

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TC (DMA2D_TypeDef * DMA2Dx)
```

#### Function description

Check if the DMA2D Transfer Complete interrupt source is enabled or disabled.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR TCIE LL\_DMA2D\_IsEnabledIT\_TC

**LL\_DMA2D\_IsEnabledIT\_TE**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_DMA2D\_IsEnabledIT\_TE (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

Check if the DMA2D Transfer Error interrupt source is enabled or disabled.

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR TEIE LL\_DMA2D\_IsEnabledIT\_TE

**LL\_DMA2D\_DeInit**

### Function name

**ErrorStatus LL\_DMA2D\_DeInit (DMA2D\_TypeDef \* DMA2Dx)**

### Function description

De-initialize DMA2D registers (registers restored to their default values).

### Parameters

- **DMA2Dx:** DMA2D Instance

### Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are de-initialized
  - ERROR: DMA2D registers are not de-initialized

**LL\_DMA2D\_Init**

### Function name

**ErrorStatus LL\_DMA2D\_Init (DMA2D\_TypeDef \* DMA2Dx, LL\_DMA2D\_InitTypeDef \* DMA2D\_InitStruct)**

### Function description

Initialize DMA2D registers according to the specified parameters in DMA2D\_InitStruct.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D\_InitStruct:** pointer to a LL\_DMA2D\_InitTypeDef structure that contains the configuration information for the specified DMA2D peripheral.

### Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are initialized according to DMA2D\_InitStruct content
  - ERROR: Issue occurred during DMA2D registers initialization

### Notes

- DMA2D transfers must be disabled to set initialization bits in configuration registers, otherwise ERROR result is returned.

### LL\_DMA2D\_StructInit

#### Function name

```
void LL_DMA2D_StructInit (LL_DMA2D_InitTypeDef * DMA2D_InitStruct)
```

#### Function description

Set each LL\_DMA2D\_InitTypeDef field to default value.

#### Parameters

- **DMA2D\_InitStruct:** pointer to a LL\_DMA2D\_InitTypeDef structure whose fields will be set to default values.

### Return values

- **None:**

### LL\_DMA2D\_ConfigLayer

#### Function name

```
void LL_DMA2D_ConfigLayer (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg, uint32_t LayerIdx)
```

#### Function description

Configure the foreground or background according to the specified parameters in the LL\_DMA2D\_LayerCfgTypeDef structure.

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D\_LayerCfg:** pointer to a LL\_DMA2D\_LayerCfgTypeDef structure that contains the configuration information for the specified layer.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

### Return values

- **None:**

### LL\_DMA2D\_LayerCfgStructInit

#### Function name

```
void LL_DMA2D_LayerCfgStructInit (LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg)
```

#### Function description

Set each LL\_DMA2D\_LayerCfgTypeDef field to default value.

#### Parameters

- **DMA2D\_LayerCfg:** pointer to a LL\_DMA2D\_LayerCfgTypeDef structure whose fields will be set to default values.

### Return values

- **None:**

### LL\_DMA2D\_ConfigOutputColor

#### Function name

```
void LL_DMA2D_ConfigOutputColor (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_ColorTypeDef *
DMA2D_ColorStruct)
```

#### Function description

Initialize DMA2D output color register according to the specified parameters in DMA2D\_ColorStruct.

#### Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D\_ColorStruct:** pointer to a LL\_DMA2D\_ColorTypeDef structure that contains the color configuration information for the specified DMA2D peripheral.

#### Return values

- **None:**

### LL\_DMA2D\_GetOutputBlueColor

#### Function name

```
uint32_t LL_DMA2D_GetOutputBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)
```

#### Function description

Return DMA2D output Blue color.

#### Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB565
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444

#### Return values

- **Output:** Blue color value between Min\_Data=0 and Max\_Data=0xFF

### LL\_DMA2D\_GetOutputGreenColor

#### Function name

```
uint32_t LL_DMA2D_GetOutputGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)
```

#### Function description

Return DMA2D output Green color.

#### Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB565
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444

### Return values

- **Output:** Green color value between Min\_Data=0 and Max\_Data=0xFF

#### LL\_DMA2D\_GetOutputRedColor

### Function name

uint32\_t LL\_DMA2D\_GetOutputRedColor (DMA2D\_TypeDef \* DMA2Dx, uint32\_t ColorMode)

### Function description

Return DMA2D output Red color.

### Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB565
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444

### Return values

- **Output:** Red color value between Min\_Data=0 and Max\_Data=0xFF

#### LL\_DMA2D\_GetOutputAlphaColor

### Function name

uint32\_t LL\_DMA2D\_GetOutputAlphaColor (DMA2D\_TypeDef \* DMA2Dx, uint32\_t ColorMode)

### Function description

Return DMA2D output Alpha color.

### Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB888
  - LL\_DMA2D\_OUTPUT\_MODE\_RGB565
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555
  - LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444

### Return values

- **Output:** Alpha color value between Min\_Data=0 and Max\_Data=0xFF

#### LL\_DMA2D\_ConfigSize

### Function name

void LL\_DMA2D\_ConfigSize (DMA2D\_TypeDef \* DMA2Dx, uint32\_t NbrOfLines, uint32\_t NbrOfPixelsPerLines)

### Function description

Configure DMA2D transfer size.

### Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min\_Data=0 and Max\_Data=0xFFFF
- **NbrOfPixelsPerLines:** Value between Min\_Data=0 and Max\_Data=0x3FFF

**Return values**

- **None:**

## 78.3 DMA2D Firmware driver defines

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

### 78.3.1 DMA2D

DMA2D

#### ***Alpha Mode***

#### LL\_DMA2D\_ALPHA\_MODE\_NO\_MODIF

No modification of the alpha channel value

#### LL\_DMA2D\_ALPHA\_MODE\_REPLACE

Replace original alpha channel value by programmed alpha value

#### LL\_DMA2D\_ALPHA\_MODE\_COMBINE

Replace original alpha channel value by programmed alpha value with original alpha channel value

#### ***CLUT Color Mode***

#### LL\_DMA2D\_CLUT\_COLOR\_MODE\_ARGB8888

ARGB8888

#### LL\_DMA2D\_CLUT\_COLOR\_MODE\_RGB888

RGB888

#### ***Get Flags Defines***

#### LL\_DMA2D\_FLAG\_CEIF

Configuration Error Interrupt Flag

#### LL\_DMA2D\_FLAG\_CTCIF

CLUT Transfer Complete Interrupt Flag

#### LL\_DMA2D\_FLAG\_CAEIF

CLUT Access Error Interrupt Flag

#### LL\_DMA2D\_FLAG\_TWIF

Transfer Watermark Interrupt Flag

#### LL\_DMA2D\_FLAG\_TCIF

Transfer Complete Interrupt Flag

#### LL\_DMA2D\_FLAG\_TEIF

Transfer Error Interrupt Flag

#### ***Input Color Mode***

#### LL\_DMA2D\_INPUT\_MODE\_ARGB8888

ARGB8888

#### LL\_DMA2D\_INPUT\_MODE\_RGB888

RGB888

#### LL\_DMA2D\_INPUT\_MODE\_RGB565

RGB565

**LL\_DMA2D\_INPUT\_MODE\_ARGB1555**

ARGB1555

**LL\_DMA2D\_INPUT\_MODE\_ARGB4444**

ARGB4444

**LL\_DMA2D\_INPUT\_MODE\_L8**

L8

**LL\_DMA2D\_INPUT\_MODE\_AL44**

AL44

**LL\_DMA2D\_INPUT\_MODE\_AL88**

AL88

**LL\_DMA2D\_INPUT\_MODE\_L4**

L4

**LL\_DMA2D\_INPUT\_MODE\_A8**

A8

**LL\_DMA2D\_INPUT\_MODE\_A4**

A4

***IT Defines***

**LL\_DMA2D\_IT\_CEIE**

Configuration Error Interrupt

**LL\_DMA2D\_IT\_CTCIE**

CLUT Transfer Complete Interrupt

**LL\_DMA2D\_IT\_CAEIE**

CLUT Access Error Interrupt

**LL\_DMA2D\_IT\_TWIE**

Transfer Watermark Interrupt

**LL\_DMA2D\_IT\_TCIE**

Transfer Complete Interrupt

**LL\_DMA2D\_IT\_TEIE**

Transfer Error Interrupt

***Mode***

**LL\_DMA2D\_MODE\_M2M**

DMA2D memory to memory transfer mode

**LL\_DMA2D\_MODE\_M2M\_PFC**

DMA2D memory to memory with pixel format conversion transfer mode

**LL\_DMA2D\_MODE\_M2M\_BLEND**

DMA2D memory to memory with blending transfer mode

**LL\_DMA2D\_MODE\_R2M**

DMA2D register to memory transfer mode

***Output Color Mode***

**LL\_DMA2D\_OUTPUT\_MODE\_ARGB8888**

ARGB8888

**LL\_DMA2D\_OUTPUT\_MODE\_RGB888**

RGB888

**LL\_DMA2D\_OUTPUT\_MODE\_RGB565**

RGB565

**LL\_DMA2D\_OUTPUT\_MODE\_ARGB1555**

ARGB1555

**LL\_DMA2D\_OUTPUT\_MODE\_ARGB4444**

ARGB4444

***Common Write and read registers Macros***

**LL\_DMA2D\_WriteReg**

**Description:**

- Write a value in DMA2D register.

**Parameters:**

- `__INSTANCE__`: DMA2D Instance
- `__REG__`: Register to be written
- `__VALUE__`: Value to be written in the register

**Return value:**

- None

**LL\_DMA2D\_ReadReg**

**Description:**

- Read a value in DMA2D register.

**Parameters:**

- `__INSTANCE__`: DMA2D Instance
- `__REG__`: Register to be read

**Return value:**

- Register: value



## 79 LL DMA Generic Driver

### 79.1 DMA Firmware driver registers structures

#### 79.1.1 LL\_DMA\_InitTypeDef

*LL\_DMA\_InitTypeDef* is defined in the `stm32f4xx_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 Channel*
- *uint32\_t Priority*
- *uint32\_t FIFOMode*
- *uint32\_t FIFOThreshold*
- *uint32\_t MemBurst*
- *uint32\_t PeriphBurst*

##### 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 Stream  
  
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::Channel***  
 Specifies the peripheral channel. This parameter can be a value of [DMA\\_LL\\_EC\\_CHANNEL](#)This feature can be modified afterwards using unitary function [LL\\_DMA\\_SetChannelSelection\(\)](#).
- ***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\\_SetStreamPriorityLevel\(\)](#).
- ***uint32\_t LL\_DMA\_InitTypeDef::FIFOMode***  
 Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of [DMA\\_LL\\_FIFOMODE](#)  
**Note:**
  - The Direct mode (FIFO mode disabled) cannot be used if the memory-to-memory data transfer is configured on the selected stream
 This feature can be modified afterwards using unitary functions [LL\\_DMA\\_EnableFifoMode\(\)](#) or [LL\\_DMA\\_EnableFifoMode\(\)](#) .
- ***uint32\_t LL\_DMA\_InitTypeDef::FIFOThreshold***  
 Specifies the FIFO threshold level. This parameter can be a value of [DMA\\_LL\\_EC\\_FIFOTHRESHOLD](#)This feature can be modified afterwards using unitary function [LL\\_DMA\\_SetFIFOThreshold\(\)](#).
- ***uint32\_t LL\_DMA\_InitTypeDef::MemBurst***  
 Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of [DMA\\_LL\\_EC\\_MBURST](#)  
**Note:**
  - The burst mode is possible only if the address Increment mode is enabled.
 This feature can be modified afterwards using unitary function [LL\\_DMA\\_SetMemoryBurstxfer\(\)](#).
- ***uint32\_t LL\_DMA\_InitTypeDef::PeriphBurst***  
 Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of [DMA\\_LL\\_EC\\_PBURST](#)  
**Note:**
  - The burst mode is possible only if the address Increment mode is enabled.
 This feature can be modified afterwards using unitary function [LL\\_DMA\\_SetPeriphBurstxfer\(\)](#).

## 79.2 DMA Firmware driver API description

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

## 79.2.1 Detailed description of functions

### LL\_DMA\_EnableStream

#### Function name

```
__STATIC_INLINE void LL_DMA_EnableStream (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Enable DMA stream.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR EN LL\_DMA\_EnableStream

### LL\_DMA\_DisableStream

#### Function name

```
__STATIC_INLINE void LL_DMA_DisableStream (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Disable DMA stream.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR EN LL\_DMA\_DisableStream

## LL\_DMA\_IsEnabledStream

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsEnabledStream (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Check if DMA stream is enabled or disabled.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR EN LL\_DMA\_IsEnabledStream

## LL\_DMA\_ConfigTransfer

### Function name

```
__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Configuration)
```

### Function description

Configure all parameters linked to DMA transfer.

## Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **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 or LL\_DMA\_MODE\_PFCTRL
  - 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:

- CR DIR LL\_DMA\_ConfigTransfer
- CR CIRC LL\_DMA\_ConfigTransfer
- CR PINC LL\_DMA\_ConfigTransfer
- CR MINC LL\_DMA\_ConfigTransfer
- CR PSIZE LL\_DMA\_ConfigTransfer
- CR MSIZE LL\_DMA\_ConfigTransfer
- CR PL LL\_DMA\_ConfigTransfer
- CR PFCTRL LL\_DMA\_ConfigTransfer

## LL\_DMA\_SetDataTransferDirection

### Function name

```
__STATIC_INLINE void LL_DMA_SetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Direction)
```

### Function description

Set Data transfer direction (read from peripheral or from memory).

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **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:

- CR DIR LL\_DMA\_SetDataTransferDirection

### LL\_DMA\_GetDataTransferDirection

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get Data transfer direction (read from peripheral or from memory).

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### 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:

- CR DIR LL\_DMA\_GetDataTransferDirection

## LL\_DMA\_SetMode

### Function name

```
__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Mode)
```

### Function description

Set DMA mode normal, circular or peripheral flow control.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Mode:** This parameter can be one of the following values:
  - LL\_DMA\_MODE\_NORMAL
  - LL\_DMA\_MODE\_CIRCULAR
  - LL\_DMA\_MODE\_PFCTRL

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR CIRC LL\_DMA\_SetMode
- CR PFCTRL LL\_DMA\_SetMode

## LL\_DMA\_GetMode

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Get DMA mode normal, circular or peripheral flow control.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_MODE\_NORMAL
  - LL\_DMA\_MODE\_CIRCULAR
  - LL\_DMA\_MODE\_PFCTRL

### Reference Manual to LL API cross reference:

- CR CIRC LL\_DMA\_GetMode
- CR PFCTRL LL\_DMA\_GetMode

### LL\_DMA\_SetPeriphIncMode

#### Function name

```
__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t IncrementMode)
```

#### Function description

Set Peripheral increment mode.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **IncrementMode:** This parameter can be one of the following values:
  - LL\_DMA\_PERIPH\_NOINCREMENT
  - LL\_DMA\_PERIPH\_INCREMENT

#### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR PINC LL\_DMA\_SetPeriphIncMode

### LL\_DMA\_GetPeriphIncMode

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Get Peripheral increment mode.



### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_PERIPH\_NOINCREMENT
  - LL\_DMA\_PERIPH\_INCREMENT

### Reference Manual to LL API cross reference:

- CR PINC LL\_DMA\_GetPeriphIncMode

### LL\_DMA\_SetMemoryIncMode

#### Function name

```
__STATIC_INLINE void LL_DMA_SetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t IncrementMode)
```

#### Function description

Set Memory increment mode.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **IncrementMode:** This parameter can be one of the following values:
  - LL\_DMA\_MEMORY\_NOINCREMENT
  - LL\_DMA\_MEMORY\_INCREMENT

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR MINC LL\_DMA\_SetMemoryIncMode

### LL\_DMA\_GetMemoryIncMode

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Get Memory increment mode.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_MEMORY\_NOINCREMENT
  - LL\_DMA\_MEMORY\_INCREMENT

### Reference Manual to LL API cross reference:

- CR MINC LL\_DMA\_GetMemoryIncMode

### LL\_DMA\_SetPeriphSize

### Function name

```
__STATIC_INLINE void LL_DMA_SetPeriphSize (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Size)
```

### Function description

Set Peripheral size.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Size:** 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:

- CR PSIZE LL\_DMA\_SetPeriphSize

## LL\_DMA\_GetPeriphSize

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Get Peripheral size.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### 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:

- CR PSIZE LL\_DMA\_GetPeriphSize

## LL\_DMA\_SetMemorySize

### Function name

```
__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Size)
```

### Function description

Set Memory size.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Size:** 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:**

- CR MSIZE LL\_DMA\_SetMemorySize

**LL\_DMA\_GetMemorySize**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAx, uint32_t Stream)
```

**Function description**

Get Memory size.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

**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:**

- CR MSIZE LL\_DMA\_GetMemorySize

**LL\_DMA\_SetIncOffsetSize**
**Function name**

```
__STATIC_INLINE void LL_DMA_SetIncOffsetSize (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t OffsetSize)
```

**Function description**

Set Peripheral increment offset size.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **OffsetSize:** This parameter can be one of the following values:
  - LL\_DMA\_OFFSETSIZE\_PSIZE
  - LL\_DMA\_OFFSETSIZE\_FIXEDTO4

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR PINCOS LL\_DMA\_SetIncOffsetSize

#### LL\_DMA\_GetIncOffsetSize

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetIncOffsetSize (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get Peripheral increment offset size.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_OFFSETSIZE\_PSIZE
  - LL\_DMA\_OFFSETSIZE\_FIXEDTO4

### Reference Manual to LL API cross reference:

- CR PINCOS LL\_DMA\_GetIncOffsetSize

#### LL\_DMA\_SetStreamPriorityLevel

### Function name

`__STATIC_INLINE void LL_DMA_SetStreamPriorityLevel (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Priority)`

### Function description

Set Stream priority level.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **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:

- CR PL LL\_DMA\_SetStreamPriorityLevel

### LL\_DMA\_GetStreamPriorityLevel

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetStreamPriorityLevel (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get Stream priority level.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### 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:**

- CR PL LL\_DMA\_GetStreamPriorityLevel

**LL\_DMA\_SetDataLength**

**Function name**

`__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t NbData)`

**Function description**

Set Number of data to transfer.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **NbData:** Between 0 to 0xFFFFFFFF

**Return values**

- **None:**

**Notes**

- This action has no effect if stream is enabled.

**Reference Manual to LL API cross reference:**

- NDTR NDT LL\_DMA\_SetDataLength

**LL\_DMA\_GetDataLength**

**Function name**

`__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Stream)`

**Function description**

Get Number of data to transfer.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Between:** 0 to 0xFFFFFFFF

### Notes

- Once the stream is enabled, the return value indicate the remaining bytes to be transmitted.

### Reference Manual to LL API cross reference:

- NDTR NDT LL\_DMA\_GetDataLength

### LL\_DMA\_SetChannelSelection

#### Function name

```
__STATIC_INLINE void LL_DMA_SetChannelSelection (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Channel)
```

#### Function description

Select Channel number associated to the Stream.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Channel:** This parameter can be one of the following values:
  - LL\_DMA\_CHANNEL\_0
  - 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

#### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR CHSEL LL\_DMA\_SetChannelSelection

### LL\_DMA\_GetChannelSelection

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetChannelSelection (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Get the Channel number associated to the Stream.



### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_CHANNEL\_0
  - 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

### Reference Manual to LL API cross reference:

- CR CHSEL LL\_DMA\_GetChannelSelection

### LL\_DMA\_SetMemoryBurstxfer

#### Function name

```
__STATIC_INLINE void LL_DMA_SetMemoryBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Mburst)
```

#### Function description

Set Memory burst transfer configuration.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Mburst:** This parameter can be one of the following values:
  - LL\_DMA\_MBURST\_SINGLE
  - LL\_DMA\_MBURST\_INC4
  - LL\_DMA\_MBURST\_INC8
  - LL\_DMA\_MBURST\_INC16

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR MBURST LL\_DMA\_SetMemoryBurstxfer

### LL\_DMA\_GetMemoryBurstxfer

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetMemoryBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get Memory burst transfer configuration.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_MBURST\_SINGLE
  - LL\_DMA\_MBURST\_INC4
  - LL\_DMA\_MBURST\_INC8
  - LL\_DMA\_MBURST\_INC16

### Reference Manual to LL API cross reference:

- CR MBURST LL\_DMA\_GetMemoryBurstxfer

### LL\_DMA\_SetPeriphBurstxfer

### Function name

`__STATIC_INLINE void LL_DMA_SetPeriphBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Pburst)`

### Function description

Set Peripheral burst transfer configuration.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Pburst:** This parameter can be one of the following values:
  - LL\_DMA\_PBURST\_SINGLE
  - LL\_DMA\_PBURST\_INC4
  - LL\_DMA\_PBURST\_INC8
  - LL\_DMA\_PBURST\_INC16

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR PBURST LL\_DMA\_SetPeriphBurstxfer

### LL\_DMA\_GetPeriphBurstxfer

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetPeriphBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get Peripheral burst transfer configuration.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_PBURST\_SINGLE
  - LL\_DMA\_PBURST\_INC4
  - LL\_DMA\_PBURST\_INC8
  - LL\_DMA\_PBURST\_INC16

### Reference Manual to LL API cross reference:

- CR PBURST LL\_DMA\_GetPeriphBurstxfer

## LL\_DMA\_SetCurrentTargetMem

### Function name

```
__STATIC_INLINE void LL_DMA_SetCurrentTargetMem (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t CurrentMemory)
```

### Function description

Set Current target (only in double buffer mode) to Memory 1 or Memory 0.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **CurrentMemory:** This parameter can be one of the following values:
  - LL\_DMA\_CURRENTTARGETMEM0
  - LL\_DMA\_CURRENTTARGETMEM1

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR CT LL\_DMA\_SetCurrentTargetMem

## LL\_DMA\_GetCurrentTargetMem

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetCurrentTargetMem (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Set Current target (only in double buffer mode) to Memory 1 or Memory 0.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_CURRENTTARGETMEM0
  - LL\_DMA\_CURRENTTARGETMEM1

**Reference Manual to LL API cross reference:**

- CR CT LL\_DMA\_GetCurrentTargetMem

**LL\_DMA\_EnableDoubleBufferMode**

**Function name**

`__STATIC_INLINE void LL_DMA_EnableDoubleBufferMode (DMA_TypeDef * DMAx, uint32_t Stream)`

**Function description**

Enable the double buffer mode.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR DBM LL\_DMA\_EnableDoubleBufferMode

**LL\_DMA\_DisableDoubleBufferMode**

**Function name**

`__STATIC_INLINE void LL_DMA_DisableDoubleBufferMode (DMA_TypeDef * DMAx, uint32_t Stream)`

**Function description**

Disable the double buffer mode.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR DBM LL\_DMA\_DisableDoubleBufferMode

## LL\_DMA\_GetFIFOStatus

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetFIFOStatus (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Get FIFO status.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_FIFOSTATUS\_0\_25
  - LL\_DMA\_FIFOSTATUS\_25\_50
  - LL\_DMA\_FIFOSTATUS\_50\_75
  - LL\_DMA\_FIFOSTATUS\_75\_100
  - LL\_DMA\_FIFOSTATUS\_EMPTY
  - LL\_DMA\_FIFOSTATUS\_FULL

### Reference Manual to LL API cross reference:

- FCR FS LL\_DMA\_GetFIFOStatus

## LL\_DMA\_DisableFifoMode

### Function name

```
__STATIC_INLINE void LL_DMA_DisableFifoMode (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Disable Fifo mode.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- FCR DMDIS LL\_DMA\_DisableFifoMode

#### LL\_DMA\_EnableFifoMode

#### Function name

```
__STATIC_INLINE void LL_DMA_EnableFifoMode (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Enable Fifo mode.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- FCR DMDIS LL\_DMA\_EnableFifoMode

#### LL\_DMA\_SetFIFOThreshold

#### Function name

```
__STATIC_INLINE void LL_DMA_SetFIFOThreshold (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Threshold)
```

#### Function description

Select FIFO threshold.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Threshold:** This parameter can be one of the following values:
  - LL\_DMA\_FIFOTHRESHOLD\_1\_4
  - LL\_DMA\_FIFOTHRESHOLD\_1\_2
  - LL\_DMA\_FIFOTHRESHOLD\_3\_4
  - LL\_DMA\_FIFOTHRESHOLD\_FULL

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FCR FTH LL\_DMA\_SetFIFOThreshold

#### LL\_DMA\_GetFIFOThreshold

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetFIFOThreshold (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get FIFO threshold.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Returned:** value can be one of the following values:
  - LL\_DMA\_FIFOTHRESHOLD\_1\_4
  - LL\_DMA\_FIFOTHRESHOLD\_1\_2
  - LL\_DMA\_FIFOTHRESHOLD\_3\_4
  - LL\_DMA\_FIFOTHRESHOLD\_FULL

### Reference Manual to LL API cross reference:

- FCR FTH LL\_DMA\_GetFIFOThreshold



## LL\_DMA\_ConfigFifo

### Function name

```
__STATIC_INLINE void LL_DMA_ConfigFifo (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t FifoMode, uint32_t FifoThreshold)
```

### Function description

Configure the FIFO .

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **FifoMode:** This parameter can be one of the following values:
  - LL\_DMA\_FIFOMODE\_ENABLE
  - LL\_DMA\_FIFOMODE\_DISABLE
- **FifoThreshold:** This parameter can be one of the following values:
  - LL\_DMA\_FIFOTHRESHOLD\_1\_4
  - LL\_DMA\_FIFOTHRESHOLD\_1\_2
  - LL\_DMA\_FIFOTHRESHOLD\_3\_4
  - LL\_DMA\_FIFOTHRESHOLD\_FULL

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FCR FTH LL\_DMA\_ConfigFifo
- FCR DMDIS LL\_DMA\_ConfigFifo

## LL\_DMA\_ConfigAddresses

### Function name

```
__STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)
```

### Function description

Configure the Source and Destination addresses.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **SrcAddress:** Between 0 to 0xFFFFFFFF
- **DstAddress:** Between 0 to 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 stream is enabled.

### Reference Manual to LL API cross reference:

- M0AR M0A LL\_DMA\_ConfigAddresses
- PAR PA LL\_DMA\_ConfigAddresses

### LL\_DMA\_SetMemoryAddress

#### Function name

```
__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t MemoryAddress)
```

#### Function description

Set the Memory address.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **MemoryAddress:** Between 0 to 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:

- M0AR M0A `LL_DMA_SetMemoryAddress`

### `LL_DMA_SetPeriphAddress`

#### Function name

```
__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t PeriphAddress)
```

#### Function description

Set the Peripheral address.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - `LL_DMA_STREAM_0`
  - `LL_DMA_STREAM_1`
  - `LL_DMA_STREAM_2`
  - `LL_DMA_STREAM_3`
  - `LL_DMA_STREAM_4`
  - `LL_DMA_STREAM_5`
  - `LL_DMA_STREAM_6`
  - `LL_DMA_STREAM_7`
- **PeriphAddress:** Between 0 to `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:

- PAR PA `LL_DMA_SetPeriphAddress`

### `LL_DMA_GetMemoryAddress`

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Get the Memory address.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Between:** 0 to 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:

- M0AR M0A LL\_DMA\_GetMemoryAddress

### LL\_DMA\_GetPeriphAddress

### Function name

`__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Get the Peripheral address.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Between:** 0 to 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:

- PAR PA LL\_DMA\_GetPeriphAddress

## LL\_DMA\_SetM2MSrcAddress

### Function name

```
__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t MemoryAddress)
```

### Function description

Set the Memory to Memory Source address.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **MemoryAddress:** Between 0 to 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:

- PAR PA LL\_DMA\_SetM2MSrcAddress

## LL\_DMA\_SetM2MDstAddress

### Function name

```
__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t MemoryAddress)
```

### Function description

Set the Memory to Memory Destination address.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **MemoryAddress:** Between 0 to 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:

- M0AR M0A LL\_DMA\_SetM2MDstAddress

### LL\_DMA\_GetM2MSrcAddress

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Get the Memory to Memory Source address.

#### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

#### Return values

- **Between:** 0 to 0xFFFFFFFF

### Notes

- Interface used for direction LL\_DMA\_DIRECTION\_MEMORY\_TO\_MEMORY only.

### Reference Manual to LL API cross reference:

- PAR PA LL\_DMA\_GetM2MSrcAddress

### LL\_DMA\_GetM2MDstAddress

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Stream)
```

#### Function description

Get the Memory to Memory Destination address.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Between:** 0 to 0xFFFFFFFF

### Notes

- Interface used for direction LL\_DMA\_DIRECTION\_MEMORY\_TO\_MEMORY only.

### Reference Manual to LL API cross reference:

- M0AR M0A LL\_DMA\_GetM2MDstAddress

#### LL\_DMA\_SetMemory1Address

### Function name

```
__STATIC_INLINE void LL_DMA_SetMemory1Address (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Address)
```

### Function description

Set Memory 1 address (used in case of Double buffer mode).

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **Address:** Between 0 to 0xFFFFFFFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- M1AR M1A LL\_DMA\_SetMemory1Address

#### LL\_DMA\_GetMemory1Address

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_GetMemory1Address (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Get Memory 1 address (used in case of Double buffer mode).

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **Between:** 0 to 0xFFFFFFFF

### Reference Manual to LL API cross reference:

- M1AR M1A LL\_DMA\_GetMemory1Address

### LL\_DMA\_IsActiveFlag\_HT0

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT0 (DMA_TypeDef * DMAx)
```

### Function description

Get Stream 0 half transfer flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- LISR HTIF0 LL\_DMA\_IsActiveFlag\_HT0

### LL\_DMA\_IsActiveFlag\_HT1

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)
```

### Function description

Get Stream 1 half transfer flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- LISR 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 Stream 2 half transfer flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR 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 Stream 3 half transfer flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR 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 Stream 4 half transfer flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR 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 Stream 5 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR HTIF0 LL\_DMA\_IsActiveFlag\_HT5

**LL\_DMA\_IsActiveFlag\_HT6**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)
```

**Function description**

Get Stream 6 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR 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 Stream 7 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR HTIF7 LL\_DMA\_IsActiveFlag\_HT7

**LL\_DMA\_IsActiveFlag\_TC0**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC0 (DMA_TypeDef * DMAx)
```

**Function description**

Get Stream 0 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR TCIF0 LL\_DMA\_IsActiveFlag\_TC0

**LL\_DMA\_IsActiveFlag\_TC1**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)
```

**Function description**

Get Stream 1 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR 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 Stream 2 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR 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 Stream 3 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR 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 Stream 4 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR 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 Stream 5 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR TCIF0 LL\_DMA\_IsActiveFlag\_TC5

**LL\_DMA\_IsActiveFlag\_TC6**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_DMA\_IsActiveFlag\_TC6 (DMA\_TypeDef \* DMAx)**

**Function description**

Get Stream 6 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR 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 Stream 7 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR TCIF7 LL\_DMA\_IsActiveFlag\_TC7

#### LL\_DMA\_IsActiveFlag\_TE0

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE0 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 0 transfer error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR TEIF0 LL\_DMA\_IsActiveFlag\_TE0

#### LL\_DMA\_IsActiveFlag\_TE1

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 1 transfer error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR 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 Stream 2 transfer error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR 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 Stream 3 transfer error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR 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 Stream 4 transfer error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR 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 Stream 5 transfer error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR TEIF0 LL\_DMA\_IsActiveFlag\_TE5

### LL\_DMA\_IsActiveFlag\_TE6

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 6 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR 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 Stream 7 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR TEIF7 LL\_DMA\_IsActiveFlag\_TE7

**LL\_DMA\_IsActiveFlag\_DME0**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_DMA\_IsActiveFlag\_DME0 (DMA\_TypeDef \* DMAx)**

**Function description**

Get Stream 0 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR DMEIF0 LL\_DMA\_IsActiveFlag\_DME0

**LL\_DMA\_IsActiveFlag\_DME1**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_DMA\_IsActiveFlag\_DME1 (DMA\_TypeDef \* DMAx)**

**Function description**

Get Stream 1 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR DMEIF1 LL\_DMA\_IsActiveFlag\_DME1

**LL\_DMA\_IsActiveFlag\_DME2**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME2 (DMA_TypeDef * DMAx)
```

**Function description**

Get Stream 2 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR DMEIF2 LL\_DMA\_IsActiveFlag\_DME2

**LL\_DMA\_IsActiveFlag\_DME3**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME3 (DMA_TypeDef * DMAx)
```

**Function description**

Get Stream 3 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- LISR DMEIF3 LL\_DMA\_IsActiveFlag\_DME3

**LL\_DMA\_IsActiveFlag\_DME4**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME4 (DMA_TypeDef * DMAx)
```

**Function description**

Get Stream 4 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR DMEIF4 LL\_DMA\_IsActiveFlag\_DME4

**LL\_DMA\_IsActiveFlag\_DME5**
**Function name**

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME5 (DMA_TypeDef * DMAx)
```



**Function description**

Get Stream 5 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR DMEIF0 LL\_DMA\_IsActiveFlag\_DME5

**LL\_DMA\_IsActiveFlag\_DME6**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_DMA\_IsActiveFlag\_DME6 (DMA\_TypeDef \* DMAx)**

**Function description**

Get Stream 6 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR DMEIF6 LL\_DMA\_IsActiveFlag\_DME6

**LL\_DMA\_IsActiveFlag\_DME7**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_DMA\_IsActiveFlag\_DME7 (DMA\_TypeDef \* DMAx)**

**Function description**

Get Stream 7 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR DMEIF7 LL\_DMA\_IsActiveFlag\_DME7

**LL\_DMA\_IsActiveFlag\_FE0**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_DMA\_IsActiveFlag\_FE0 (DMA\_TypeDef \* DMAx)**

**Function description**

Get Stream 0 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR FEIF0 LL\_DMA\_IsActiveFlag\_FE0

#### LL\_DMA\_IsActiveFlag\_FE1

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE1 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 1 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR FEIF1 LL\_DMA\_IsActiveFlag\_FE1

#### LL\_DMA\_IsActiveFlag\_FE2

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE2 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 2 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR FEIF2 LL\_DMA\_IsActiveFlag\_FE2

#### LL\_DMA\_IsActiveFlag\_FE3

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE3 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 3 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- LISR FEIF3 LL\_DMA\_IsActiveFlag\_FE3

### LL\_DMA\_IsActiveFlag\_FE4

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE4 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 4 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR FEIF4 LL\_DMA\_IsActiveFlag\_FE4

### LL\_DMA\_IsActiveFlag\_FE5

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE5 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 5 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR FEIF0 LL\_DMA\_IsActiveFlag\_FE5

### LL\_DMA\_IsActiveFlag\_FE6

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE6 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 6 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- HISR FEIF6 LL\_DMA\_IsActiveFlag\_FE6

### LL\_DMA\_IsActiveFlag\_FE7

#### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE7 (DMA_TypeDef * DMAx)
```

#### Function description

Get Stream 7 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- HISR FEIF7 LL\_DMA\_IsActiveFlag\_FE7

**LL\_DMA\_ClearFlag\_HT0**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT0 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 0 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CHTIF0 LL\_DMA\_ClearFlag\_HT0

**LL\_DMA\_ClearFlag\_HT1**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 1 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CHTIF1 LL\_DMA\_ClearFlag\_HT1

**LL\_DMA\_ClearFlag\_HT2**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 2 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CHTIF2 LL\_DMA\_ClearFlag\_HT2

**LL\_DMA\_ClearFlag\_HT3**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 3 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CHTIF3 LL\_DMA\_ClearFlag\_HT3

**LL\_DMA\_ClearFlag\_HT4**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 4 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CHTIF4 LL\_DMA\_ClearFlag\_HT4

**LL\_DMA\_ClearFlag\_HT5**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 5 half transfer flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CHTIF5 LL\_DMA\_ClearFlag\_HT5

**LL\_DMA\_ClearFlag\_HT6**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)
```

### Function description

Clear Stream 6 half transfer flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- HIFCR CHTIF6 LL\_DMA\_ClearFlag\_HT6

**LL\_DMA\_ClearFlag\_HT7**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_HT7 (DMA\_TypeDef \* DMAx)**

### Function description

Clear Stream 7 half transfer flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- HIFCR CHTIF7 LL\_DMA\_ClearFlag\_HT7

**LL\_DMA\_ClearFlag\_TC0**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_TC0 (DMA\_TypeDef \* DMAx)**

### Function description

Clear Stream 0 transfer complete flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- LIFCR CTCIF0 LL\_DMA\_ClearFlag\_TC0

**LL\_DMA\_ClearFlag\_TC1**

### Function name

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_TC1 (DMA\_TypeDef \* DMAx)**

### Function description

Clear Stream 1 transfer complete flag.

### Parameters

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTCIF1 LL\_DMA\_ClearFlag\_TC1

**LL\_DMA\_ClearFlag\_TC2**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 2 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTCIF2 LL\_DMA\_ClearFlag\_TC2

**LL\_DMA\_ClearFlag\_TC3**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 3 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTCIF3 LL\_DMA\_ClearFlag\_TC3

**LL\_DMA\_ClearFlag\_TC4**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 4 transfer complete flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CTCIF4 LL\_DMA\_ClearFlag\_TC4

### LL\_DMA\_ClearFlag\_TC5

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 5 transfer complete flag.

#### Parameters

- **DMAx**: DMAx Instance

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- HIFCR CTCIF5 LL\_DMA\_ClearFlag\_TC5

### LL\_DMA\_ClearFlag\_TC6

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 6 transfer complete flag.

#### Parameters

- **DMAx**: DMAx Instance

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- HIFCR CTCIF6 LL\_DMA\_ClearFlag\_TC6

### LL\_DMA\_ClearFlag\_TC7

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 7 transfer complete flag.

#### Parameters

- **DMAx**: DMAx Instance

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- HIFCR CTCIF7 LL\_DMA\_ClearFlag\_TC7

### LL\_DMA\_ClearFlag\_TE0

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_TE0 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 0 transfer error flag.



**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTEIF0 LL\_DMA\_ClearFlag\_TE0

**LL\_DMA\_ClearFlag\_TE1**

**Function name**

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_TE1 (DMA\_TypeDef \* DMAx)**

**Function description**

Clear Stream 1 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTEIF1 LL\_DMA\_ClearFlag\_TE1

**LL\_DMA\_ClearFlag\_TE2**

**Function name**

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_TE2 (DMA\_TypeDef \* DMAx)**

**Function description**

Clear Stream 2 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTEIF2 LL\_DMA\_ClearFlag\_TE2

**LL\_DMA\_ClearFlag\_TE3**

**Function name**

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_TE3 (DMA\_TypeDef \* DMAx)**

**Function description**

Clear Stream 3 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CTEIF3 LL\_DMA\_ClearFlag\_TE3

**LL\_DMA\_ClearFlag\_TE4**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 4 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CTEIF4 LL\_DMA\_ClearFlag\_TE4

**LL\_DMA\_ClearFlag\_TE5**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 5 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CTEIF5 LL\_DMA\_ClearFlag\_TE5

**LL\_DMA\_ClearFlag\_TE6**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 6 transfer error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CTEIF6 LL\_DMA\_ClearFlag\_TE6

**LL\_DMA\_ClearFlag\_TE7**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)
```

### Function description

Clear Stream 7 transfer error flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- HIFCR CTEIF7 LL\_DMA\_ClearFlag\_TE7

### LL\_DMA\_ClearFlag\_DME0

### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME0 (DMA_TypeDef * DMAx)
```

### Function description

Clear Stream 0 direct mode error flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- LIFCR CDMEIF0 LL\_DMA\_ClearFlag\_DME0

### LL\_DMA\_ClearFlag\_DME1

### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME1 (DMA_TypeDef * DMAx)
```

### Function description

Clear Stream 1 direct mode error flag.

### Parameters

- **DMAx:** DMAx Instance

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- LIFCR CDMEIF1 LL\_DMA\_ClearFlag\_DME1

### LL\_DMA\_ClearFlag\_DME2

### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME2 (DMA_TypeDef * DMAx)
```

### Function description

Clear Stream 2 direct mode error flag.

### Parameters

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CDMEIF2 LL\_DMA\_ClearFlag\_DME2

**LL\_DMA\_ClearFlag\_DME3**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME3 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 3 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CDMEIF3 LL\_DMA\_ClearFlag\_DME3

**LL\_DMA\_ClearFlag\_DME4**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME4 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 4 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CDMEIF4 LL\_DMA\_ClearFlag\_DME4

**LL\_DMA\_ClearFlag\_DME5**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME5 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 5 direct mode error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CDMEIF5 LL\_DMA\_ClearFlag\_DME5

### LL\_DMA\_ClearFlag\_DME6

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME6 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 6 direct mode error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- HIFCR CDMEIF6 LL\_DMA\_ClearFlag\_DME6

### LL\_DMA\_ClearFlag\_DME7

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_DME7 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 7 direct mode error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- HIFCR CDMEIF7 LL\_DMA\_ClearFlag\_DME7

### LL\_DMA\_ClearFlag\_FE0

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_FE0 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 0 FIFO error flag.

#### Parameters

- **DMAx:** DMAx Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- LIFCR CFEIF0 LL\_DMA\_ClearFlag\_FE0

### LL\_DMA\_ClearFlag\_FE1

#### Function name

```
__STATIC_INLINE void LL_DMA_ClearFlag_FE1 (DMA_TypeDef * DMAx)
```

#### Function description

Clear Stream 1 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CFEIF1 LL\_DMA\_ClearFlag\_FE1

**LL\_DMA\_ClearFlag\_FE2**

**Function name**

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_FE2 (DMA\_TypeDef \* DMAx)**

**Function description**

Clear Stream 2 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CFEIF2 LL\_DMA\_ClearFlag\_FE2

**LL\_DMA\_ClearFlag\_FE3**

**Function name**

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_FE3 (DMA\_TypeDef \* DMAx)**

**Function description**

Clear Stream 3 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- LIFCR CFEIF3 LL\_DMA\_ClearFlag\_FE3

**LL\_DMA\_ClearFlag\_FE4**

**Function name**

**\_\_STATIC\_INLINE void LL\_DMA\_ClearFlag\_FE4 (DMA\_TypeDef \* DMAx)**

**Function description**

Clear Stream 4 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CFEIF4 LL\_DMA\_ClearFlag\_FE4

**LL\_DMA\_ClearFlag\_FE5**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_FE5 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 5 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CFEIF5 LL\_DMA\_ClearFlag\_FE5

**LL\_DMA\_ClearFlag\_FE6**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_FE6 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 6 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CFEIF6 LL\_DMA\_ClearFlag\_FE6

**LL\_DMA\_ClearFlag\_FE7**
**Function name**

```
__STATIC_INLINE void LL_DMA_ClearFlag_FE7 (DMA_TypeDef * DMAx)
```

**Function description**

Clear Stream 7 FIFO error flag.

**Parameters**

- **DMAx:** DMAx Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- HIFCR CFEIF7 LL\_DMA\_ClearFlag\_FE7

**LL\_DMA\_EnableIT\_HT**
**Function name**

```
__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Enable Half transfer interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR HTIE LL\_DMA\_EnableIT\_HT

### LL\_DMA\_EnableIT\_TE

### Function name

```
__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Enable Transfer error interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR TEIE LL\_DMA\_EnableIT\_TE

### LL\_DMA\_EnableIT\_TC

### Function name

```
__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Enable Transfer complete interrupt.



**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR TCIE LL\_DMA\_EnableIT\_TC

**LL\_DMA\_EnableIT\_DME**
**Function name**

```
__STATIC_INLINE void LL_DMA_EnableIT_DME (DMA_TypeDef * DMAx, uint32_t Stream)
```

**Function description**

Enable Direct mode error interrupt.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR DMEIE LL\_DMA\_EnableIT\_DME

**LL\_DMA\_EnableIT\_FE**
**Function name**

```
__STATIC_INLINE void LL_DMA_EnableIT_FE (DMA_TypeDef * DMAx, uint32_t Stream)
```

**Function description**

Enable FIFO error interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FCR FEIE LL\_DMA\_EnableIT\_FE

### LL\_DMA\_DisableIT\_HT

### Function name

```
__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Disable Half transfer interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR HTIE LL\_DMA\_DisableIT\_HT

### LL\_DMA\_DisableIT\_TE

### Function name

```
__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Disable Transfer error interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR TEIE LL\_DMA\_DisableIT\_TE

#### LL\_DMA\_DisableIT\_TC

### Function name

```
__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Disable Transfer complete interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR TCIE LL\_DMA\_DisableIT\_TC

#### LL\_DMA\_DisableIT\_DME

### Function name

```
__STATIC_INLINE void LL_DMA_DisableIT_DME (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Disable Direct mode error interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR DMEIE LL\_DMA\_DisableIT\_DME

### LL\_DMA\_DisableIT\_FE

### Function name

```
__STATIC_INLINE void LL_DMA_DisableIT_FE (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Disable FIFO error interrupt.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- FCR FEIE LL\_DMA\_DisableIT\_FE

### LL\_DMA\_IsEnabledIT\_HT

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Check if Half transfer interrupt is enabled.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR HTIE LL\_DMA\_IsEnabledIT\_HT

### LL\_DMA\_IsEnabledIT\_TE

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Check if Transfer error nterrupt is enabled.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR TEIE LL\_DMA\_IsEnabledIT\_TE

### LL\_DMA\_IsEnabledIT\_TC

### Function name

```
__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Stream)
```

### Function description

Check if Transfer complete interrupt is enabled.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR TCIE LL\_DMA\_IsEnabledIT\_TC

#### LL\_DMA\_IsEnabledIT\_DME

### Function name

`__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_DME (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Check if Direct mode error interrupt is enabled.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR DMEIE LL\_DMA\_IsEnabledIT\_DME

#### LL\_DMA\_IsEnabledIT\_FE

### Function name

`__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_FE (DMA_TypeDef * DMAx, uint32_t Stream)`

### Function description

Check if FIFO error interrupt is enabled.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- FCR FEIE LL\_DMA\_IsEnabledIT\_FE

### LL\_DMA\_Init

#### Function name

**uint32\_t LL\_DMA\_Init (DMA\_TypeDef \* DMAx, uint32\_t Stream, LL\_DMA\_InitTypeDef \* DMA\_InitStruct)**

#### Function description

Initialize the DMA registers according to the specified parameters in DMA\_InitStruct.

### Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
- **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\_Streamy Instance to DMAx Instance and Streamy, use helper macros :  
`__LL_DMA_GET_INSTANCE __LL_DMA_GET_STREAM`

### LL\_DMA\_DeInit

#### Function name

**uint32\_t LL\_DMA\_DeInit (DMA\_TypeDef \* DMAx, uint32\_t Stream)**

#### Function description

De-initialize the DMA registers to their default reset values.

**Parameters**

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL\_DMA\_STREAM\_0
  - LL\_DMA\_STREAM\_1
  - LL\_DMA\_STREAM\_2
  - LL\_DMA\_STREAM\_3
  - LL\_DMA\_STREAM\_4
  - LL\_DMA\_STREAM\_5
  - LL\_DMA\_STREAM\_6
  - LL\_DMA\_STREAM\_7
  - LL\_DMA\_STREAM\_ALL

**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:**

## 79.3 DMA Firmware driver defines

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

### 79.3.1 DMA DMA CHANNEL

LL\_DMA\_CHANNEL\_0

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



**CURRENTTARGETMEM**

**LL\_DMA\_CURRENTTARGETMEM0**

Set CurrentTarget Memory to Memory 0

**LL\_DMA\_CURRENTTARGETMEM1**

Set CurrentTarget Memory to Memory 1

**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

**DOUBLEBUFFER MODE**

**LL\_DMA\_DOUBLEBUFFER\_MODE\_DISABLE**

Disable double buffering mode

**LL\_DMA\_DOUBLEBUFFER\_MODE\_ENABLE**

Enable double buffering mode

**FIFOSTATUS 0**

**LL\_DMA\_FIFOSTATUS\_0\_25**

$0 < \text{fifo\_level} < 1/4$

**LL\_DMA\_FIFOSTATUS\_25\_50**

$1/4 < \text{fifo\_level} < 1/2$

**LL\_DMA\_FIFOSTATUS\_50\_75**

$1/2 < \text{fifo\_level} < 3/4$

**LL\_DMA\_FIFOSTATUS\_75\_100**

$3/4 < \text{fifo\_level} < \text{full}$

**LL\_DMA\_FIFOSTATUS\_EMPTY**

FIFO is empty

**LL\_DMA\_FIFOSTATUS\_FULL**

FIFO is full

**FIFOTHRESHOLD**

**LL\_DMA\_FIFOTHRESHOLD\_1\_4**

FIFO threshold 1 quart full configuration

**LL\_DMA\_FIFOTHRESHOLD\_1\_2**

FIFO threshold half full configuration

**LL\_DMA\_FIFOTHRESHOLD\_3\_4**

FIFO threshold 3 quarts full configuration

**LL\_DMA\_FIFOTHRESHOLD\_FULL**

FIFO threshold full configuration

**MBURST**

**LL\_DMA\_MBURST\_SINGLE**

Memory burst single transfer configuration

**LL\_DMA\_MBURST\_INC4**

Memory burst of 4 beats transfer configuration

**LL\_DMA\_MBURST\_INC8**

Memory burst of 8 beats transfer configuration

**LL\_DMA\_MBURST\_INC16**

Memory burst of 16 beats transfer configuration

**MDATAALIGN****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****LL\_DMA\_MEMORY\_NOINCREMENT**

Memory increment mode Disable

**LL\_DMA\_MEMORY\_INCREMENT**

Memory increment mode Enable

**MODE****LL\_DMA\_MODE\_NORMAL**

Normal Mode

**LL\_DMA\_MODE\_CIRCULAR**

Circular Mode

**LL\_DMA\_MODE\_PFCTRL**

Peripheral flow control mode

**OFFSETSIZE****LL\_DMA\_OFFSETSIZE\_PSIZE**

Peripheral increment offset size is linked to the PSIZE

**LL\_DMA\_OFFSETSIZE\_FIXEDTO4**

Peripheral increment offset size is fixed to 4 (32-bit alignment)

**PBURST****LL\_DMA\_PBURST\_SINGLE**

Peripheral burst single transfer configuration

**LL\_DMA\_PBURST\_INC4**

Peripheral burst of 4 beats transfer configuration

**LL\_DMA\_PBURST\_INC8**

Peripheral burst of 8 beats transfer configuration

**LL\_DMA\_PBURST\_INC16**

Peripheral burst of 16 beats transfer configuration

**PDATAALIGN****LL\_DMA\_PDATAALIGN\_BYTE**

Peripheral data alignment : Byte

**LL\_DMA\_PDATAALIGN\_HALFWORD**

Peripheral data alignment : HalfWord

**LL\_DMA\_PDATAALIGN\_WORD**

Peripheral data alignment : Word

**PERIPH****LL\_DMA\_PERIPH\_NOINCREMENT**

Peripheral increment mode Disable

**LL\_DMA\_PERIPH\_INCREMENT**

Peripheral increment mode Enable

**PRIORITY****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

**STREAM****LL\_DMA\_STREAM\_0****LL\_DMA\_STREAM\_1****LL\_DMA\_STREAM\_2****LL\_DMA\_STREAM\_3****LL\_DMA\_STREAM\_4****LL\_DMA\_STREAM\_5****LL\_DMA\_STREAM\_6****LL\_DMA\_STREAM\_7****LL\_DMA\_STREAM\_ALL****Convert DMAxStreamy**

### **\_\_LL\_DMA\_GET\_INSTANCE**

**Description:**

- Convert DMAx\_Streamy into DMAx.

**Parameters:**

- `__STREAM_INSTANCE__`: DMAx\_Streamy

**Return value:**

- DMAx

### **\_\_LL\_DMA\_GET\_STREAM**

**Description:**

- Convert DMAx\_Streamy into LL\_DMA\_STREAM\_y.

**Parameters:**

- `__STREAM_INSTANCE__`: DMAx\_Streamy

**Return value:**

- LL\_DMA\_CHANNEL\_y

### **\_\_LL\_DMA\_GET\_STREAM\_INSTANCE**

**Description:**

- Convert DMA Instance DMAx and LL\_DMA\_STREAM\_y into DMAx\_Streamy.

**Parameters:**

- `__DMA_INSTANCE__`: DMAx
- `__STREAM__`: LL\_DMA\_STREAM\_y

**Return value:**

- DMAx\_Streamy

**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

**DMA\_LL\_FIFOMODE**

#### **LL\_DMA\_FIFOMODE\_DISABLE**

FIFO mode disable (direct mode is enabled)

**LL\_DMA\_FIFOMODE\_ENABLE**

FIFO mode enable

## 80 LL FMPI2C Generic Driver

### 80.1 FMPI2C Firmware driver registers structures

#### 80.1.1 LL\_FMPI2C\_InitTypeDef

*LL\_FMPI2C\_InitTypeDef* is defined in the `stm32f4xx_ll_fmapi2c.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\_FMPI2C\_InitTypeDef::PeripheralMode*  
Specifies the peripheral mode. This parameter can be a value of [FMPI2C\\_LL\\_EC\\_PERIPHERAL\\_MODE](#)This feature can be modified afterwards using unitary function `LL_FMPI2C_SetMode()`.
- *uint32\_t LL\_FMPI2C\_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_FMPI2C_CONVERT_TIMINGS()`This feature can be modified afterwards using unitary function `LL_FMPI2C_SetTiming()`.
- *uint32\_t LL\_FMPI2C\_InitTypeDef::AnalogFilter*  
Enables or disables analog noise filter. This parameter can be a value of [FMPI2C\\_LL\\_EC\\_ANALOGFILTER\\_SELECTION](#)This feature can be modified afterwards using unitary functions `LL_FMPI2C_EnableAnalogFilter()` or `LL_FMPI2C_DisableAnalogFilter()`.
- *uint32\_t LL\_FMPI2C\_InitTypeDef::DigitalFilter*  
Configures the digital noise filter. This parameter can be a number between `Min_Data = 0x00` and `Max_Data = 0x0F`This feature can be modified afterwards using unitary function `LL_FMPI2C_SetDigitalFilter()`.
- *uint32\_t LL\_FMPI2C\_InitTypeDef::OwnAddress1*  
Specifies the device own address 1. This parameter must be a value between `Min_Data = 0x00` and `Max_Data = 0x3FF`This feature can be modified afterwards using unitary function `LL_FMPI2C_SetOwnAddress1()`.
- *uint32\_t LL\_FMPI2C\_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 [FMPI2C\\_LL\\_EC\\_I2C\\_ACKNOWLEDGE](#)This feature can be modified afterwards using unitary function `LL_FMPI2C_AcknowledgeNextData()`.
- *uint32\_t LL\_FMPI2C\_InitTypeDef::OwnAddrSize*  
Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of [FMPI2C\\_LL\\_EC\\_OWADDRESS1](#)This feature can be modified afterwards using unitary function `LL_FMPI2C_SetOwnAddress1()`.

### 80.2 FMPI2C Firmware driver API description

The following section lists the various functions of the FMPI2C library.

#### 80.2.1 Detailed description of functions

##### LL\_FMPI2C\_Enable

###### Function name

```
__STATIC_INLINE void LL_FMPI2C_Enable (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Enable FMPI2C peripheral (PE = 1).

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- CR1 PE LL\_FMPI2C\_Enable

### LL\_FMPI2C\_Disable

### Function name

```
__STATIC_INLINE void LL_FMPI2C_Disable (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Disable FMPI2C peripheral (PE = 0).

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Notes

- When PE = 0, the FMPI2C 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\_FMPI2C\_Disable

### LL\_FMPI2C\_IsEnabled

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabled (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Check if the FMPI2C peripheral is enabled or disabled.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **State**: of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR1 PE LL\_FMPI2C\_IsEnabled

### LL\_FMPI2C\_ConfigFilters

### Function name

```
__STATIC_INLINE void LL_FMPI2C_ConfigFilters (FMPI2C_TypeDef * FMPI2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)
```

### Function description

Configure Noise Filters (Analog and Digital).

### Parameters

- **FMPI2Cx**: FMPI2C Instance.
- **AnalogFilter**: This parameter can be one of the following values:
  - LL\_FMPI2C\_ANALOGFILTER\_ENABLE
  - LL\_FMPI2C\_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\*tfmpi2cclk). 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]\*tfmpi2cclk.

### 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 FMPI2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- CR1 ANFOFF LL\_FMPI2C\_ConfigFilters
- CR1 DNF LL\_FMPI2C\_ConfigFilters

#### LL\_FMPI2C\_SetDigitalFilter

### Function name

**\_\_STATIC\_INLINE void LL\_FMPI2C\_SetDigitalFilter (FMPI2C\_TypeDef \* FMPI2Cx, uint32\_t DigitalFilter)**

### Function description

Configure Digital Noise Filter.

### Parameters

- **FMPI2Cx**: FMPI2C 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\*tfmpi2cclk). 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]\*tfmpi2cclk.

### 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 FMPI2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- CR1 DNF LL\_FMPI2C\_SetDigitalFilter

#### LL\_FMPI2C\_GetDigitalFilter

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_FMPI2C\_GetDigitalFilter (FMPI2C\_TypeDef \* FMPI2Cx)**

### Function description

Get the current Digital Noise Filter configuration.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.



**Return values**

- **Value:** between Min\_Data=0x0 and Max\_Data=0xF

**Reference Manual to LL API cross reference:**

- CR1 DNF LL\_FMPI2C\_GetDigitalFilter

**LL\_FMPI2C\_EnableAnalogFilter**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableAnalogFilter (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable Analog Noise Filter.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- This filter can only be programmed when the FMPI2C is disabled (PE = 0).

**Reference Manual to LL API cross reference:**

- CR1 ANFOFF LL\_FMPI2C\_EnableAnalogFilter

**LL\_FMPI2C\_DisableAnalogFilter**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableAnalogFilter (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable Analog Noise Filter.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- This filter can only be programmed when the FMPI2C is disabled (PE = 0).

**Reference Manual to LL API cross reference:**

- CR1 ANFOFF LL\_FMPI2C\_DisableAnalogFilter

**LL\_FMPI2C\_IsEnabledAnalogFilter**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledAnalogFilter (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if Analog Noise Filter is enabled or disabled.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR1 ANFOFF LL\_FMPI2C\_IsEnabledAnalogFilter

#### LL\_FMPI2C\_EnableDMAReq\_TX

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableDMAReq_TX (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable DMA transmission requests.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL\_FMPI2C\_EnableDMAReq\_TX

#### LL\_FMPI2C\_DisableDMAReq\_TX

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableDMAReq_TX (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable DMA transmission requests.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL\_FMPI2C\_DisableDMAReq\_TX

#### LL\_FMPI2C\_IsEnabledDMAReq\_TX

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledDMAReq_TX (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if DMA transmission requests are enabled or disabled.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL\_FMPI2C\_IsEnabledDMAReq\_TX

### LL\_FMPI2C\_EnableDMAReq\_RX

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableDMAReq_RX (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable DMA reception requests.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL\_FMPI2C\_EnableDMAReq\_RX

### LL\_FMPI2C\_DisableDMAReq\_RX

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableDMAReq_RX (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable DMA reception requests.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL\_FMPI2C\_DisableDMAReq\_RX

### LL\_FMPI2C\_IsEnabledDMAReq\_RX

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledDMAReq_RX (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if DMA reception requests are enabled or disabled.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL\_FMPI2C\_IsEnabledDMAReq\_RX

### LL\_FMPI2C\_DMA\_GetRegAddr

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_DMA_GetRegAddr (FMPI2C_TypeDef * FMPI2Cx, uint32_t Direction)
```

### Function description

Get the data register address used for DMA transfer.

### Parameters

- **FMPI2Cx:** FMPI2C Instance
- **Direction:** This parameter can be one of the following values:
  - LL\_FMPI2C\_DMA\_REG\_DATA\_TRANSMIT
  - LL\_FMPI2C\_DMA\_REG\_DATA\_RECEIVE

### Return values

- **Address:** of data register

### Reference Manual to LL API cross reference:

- TXDR TXDATA LL\_FMPI2C\_DMA\_GetRegAddr
- RXDR RXDATA LL\_FMPI2C\_DMA\_GetRegAddr

### LL\_FMPI2C\_EnableClockStretching

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableClockStretching (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable Clock stretching.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **None:**

#### Notes

- This bit can only be programmed when the FMPI2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL\_FMPI2C\_EnableClockStretching

### LL\_FMPI2C\_DisableClockStretching

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableClockStretching (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable Clock stretching.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **None:**

#### Notes

- This bit can only be programmed when the FMPI2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL\_FMPI2C\_DisableClockStretching

### LL\_FMPI2C\_IsEnabledClockStretching

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledClockStretching (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if Clock stretching is enabled or disabled.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL\_FMPI2C\_IsEnabledClockStretching

### LL\_FMPI2C\_EnableSlaveByteControl

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableSlaveByteControl (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable hardware byte control in slave mode.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 SBC LL\_FMPI2C\_EnableSlaveByteControl

### LL\_FMPI2C\_DisableSlaveByteControl

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableSlaveByteControl (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable hardware byte control in slave mode.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 SBC LL\_FMPI2C\_DisableSlaveByteControl

### LL\_FMPI2C\_IsEnabledSlaveByteControl

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledSlaveByteControl (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if hardware byte control in slave mode is enabled or disabled.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **State**: of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR1 SBC LL\_FMPI2C\_IsEnabledSlaveByteControl

**LL\_FMPI2C\_EnableGeneralCall**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableGeneralCall (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable General Call.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **None**:

**Notes**

- When enabled the Address 0x00 is ACKed.

**Reference Manual to LL API cross reference:**

- CR1 GCEN LL\_FMPI2C\_EnableGeneralCall

**LL\_FMPI2C\_DisableGeneralCall**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableGeneralCall (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable General Call.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **None**:

**Notes**

- When disabled the Address 0x00 is NACKed.

**Reference Manual to LL API cross reference:**

- CR1 GCEN LL\_FMPI2C\_DisableGeneralCall

**LL\_FMPI2C\_IsEnabledGeneralCall**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledGeneralCall (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if General Call is enabled or disabled.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR1 GCEN LL\_FMPI2C\_IsEnabledGeneralCall

### LL\_FMPI2C\_SetMasterAddressingMode

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetMasterAddressingMode (FMPI2C_TypeDef * FMPI2Cx, uint32_t AddressingMode)
```

### Function description

Configure the Master to operate in 7-bit or 10-bit addressing mode.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **AddressingMode:** This parameter can be one of the following values:
  - LL\_FMPI2C\_ADDRESSING\_MODE\_7BIT
  - LL\_FMPI2C\_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\_FMPI2C\_SetMasterAddressingMode

### LL\_FMPI2C\_GetMasterAddressingMode

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetMasterAddressingMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the Master addressing mode.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_FMPI2C\_ADDRESSING\_MODE\_7BIT
  - LL\_FMPI2C\_ADDRESSING\_MODE\_10BIT

### Reference Manual to LL API cross reference:

- CR2 ADD10 LL\_FMPI2C\_GetMasterAddressingMode

### LL\_FMPI2C\_SetOwnAddress1

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetOwnAddress1 (FMPI2C_TypeDef * FMPI2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)
```

### Function description

Set the Own Address1.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_OWNADDRESS1\_7BIT
  - LL\_FMPI2C\_OWNADDRESS1\_10BIT

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- OAR1 OA1 LL\_FMPI2C\_SetOwnAddress1
- OAR1 OA1MODE LL\_FMPI2C\_SetOwnAddress1

#### LL\_FMPI2C\_EnableOwnAddress1

### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableOwnAddress1 (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Enable acknowledge on Own Address1 match address.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- OAR1 OA1EN LL\_FMPI2C\_EnableOwnAddress1

#### LL\_FMPI2C\_DisableOwnAddress1

### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableOwnAddress1 (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Disable acknowledge on Own Address1 match address.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- OAR1 OA1EN LL\_FMPI2C\_DisableOwnAddress1

#### LL\_FMPI2C\_IsEnabledOwnAddress1

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledOwnAddress1 (FMPI2C_TypeDef * FMPI2Cx)
```



### Function description

Check if Own Address1 acknowledge is enabled or disabled.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **State**: of bit (1 or 0).

### Reference Manual to LL API cross reference:

- OAR1 OA1EN LL\_FMPI2C\_IsEnabledOwnAddress1

### LL\_FMPI2C\_SetOwnAddress2

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetOwnAddress2 (FMPI2C_TypeDef * FMPI2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)
```

### Function description

Set the 7bits Own Address2.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.
- **OwnAddress2**: Value between Min\_Data=0 and Max\_Data=0x7F.
- **OwnAddrMask**: This parameter can be one of the following values:
  - LL\_FMPI2C\_OWNADDRESS2\_NOMASK
  - LL\_FMPI2C\_OWNADDRESS2\_MASK01
  - LL\_FMPI2C\_OWNADDRESS2\_MASK02
  - LL\_FMPI2C\_OWNADDRESS2\_MASK03
  - LL\_FMPI2C\_OWNADDRESS2\_MASK04
  - LL\_FMPI2C\_OWNADDRESS2\_MASK05
  - LL\_FMPI2C\_OWNADDRESS2\_MASK06
  - LL\_FMPI2C\_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\_FMPI2C\_SetOwnAddress2
- OAR2 OA2MSK LL\_FMPI2C\_SetOwnAddress2

### LL\_FMPI2C\_EnableOwnAddress2

### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableOwnAddress2 (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Enable acknowledge on Own Address2 match address.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- OAR2 OA2EN LL\_FMPI2C\_EnableOwnAddress2

### LL\_FMPI2C\_DisableOwnAddress2

### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableOwnAddress2 (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Disable acknowledge on Own Address2 match address.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- OAR2 OA2EN LL\_FMPI2C\_DisableOwnAddress2

### LL\_FMPI2C\_IsEnabledOwnAddress2

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledOwnAddress2 (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Check if Own Address1 acknowledge is enabled or disabled.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- OAR2 OA2EN LL\_FMPI2C\_IsEnabledOwnAddress2

### LL\_FMPI2C\_SetTiming

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetTiming (FMPI2C_TypeDef * FMPI2Cx, uint32_t Timing)
```

### Function description

Configure the SDA setup, hold time and the SCL high, low period.

### Parameters

- **FMPI2Cx:** FMPI2C 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 FMPI2C is disabled (PE = 0).
- This parameter is computed with the STM32CubeMX Tool.

**Reference Manual to LL API cross reference:**

- TIMINGR TIMINGR LL\_FMPI2C\_SetTiming

**LL\_FMPI2C\_GetTimingPrescaler**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetTimingPrescaler (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Get the Timing Prescaler setting.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **Value:** between Min\_Data=0x0 and Max\_Data=0xF

**Reference Manual to LL API cross reference:**

- TIMINGR PRESC LL\_FMPI2C\_GetTimingPrescaler

**LL\_FMPI2C\_GetClockLowPeriod**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetClockLowPeriod (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Get the SCL low period setting.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **Value:** between Min\_Data=0x00 and Max\_Data=0xFF

**Reference Manual to LL API cross reference:**

- TIMINGR SCLL LL\_FMPI2C\_GetClockLowPeriod

**LL\_FMPI2C\_GetClockHighPeriod**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetClockHighPeriod (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Get the SCL high period setting.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **Value:** between Min\_Data=0x00 and Max\_Data=0xFF

**Reference Manual to LL API cross reference:**

- TIMINGR SCLH LL\_FMPI2C\_GetClockHighPeriod

**LL\_FMPI2C\_GetDataHoldTime**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetDataHoldTime (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the SDA hold time.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Value:** between Min\_Data=0x0 and Max\_Data=0xF

### Reference Manual to LL API cross reference:

- TIMINGR SDADEL LL\_FMPI2C\_GetDataHoldTime

### LL\_FMPI2C\_GetDataSetupTime

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetDataSetupTime (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the SDA setup time.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Value:** between Min\_Data=0x0 and Max\_Data=0xF

### Reference Manual to LL API cross reference:

- TIMINGR SCLDEL LL\_FMPI2C\_GetDataSetupTime

### LL\_FMPI2C\_SetMode

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetMode (FMPI2C_TypeDef * FMPI2Cx, uint32_t PeripheralMode)
```

### Function description

Configure peripheral mode.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL\_FMPI2C\_MODE\_I2C
  - LL\_FMPI2C\_MODE\_SMBUS\_HOST
  - LL\_FMPI2C\_MODE\_SMBUS\_DEVICE
  - LL\_FMPI2C\_MODE\_SMBUS\_DEVICE\_ARP

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- CR1 SMBHEN LL\_FMPI2C\_SetMode
- CR1 SMBDEN LL\_FMPI2C\_SetMode

## LL\_FMPI2C\_GetMode

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get peripheral mode.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_FMPI2C\_MODE\_I2C
  - LL\_FMPI2C\_MODE\_SMBUS\_HOST
  - LL\_FMPI2C\_MODE\_SMBUS\_DEVICE
  - LL\_FMPI2C\_MODE\_SMBUS\_DEVICE\_ARP

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- CR1 SMBHEN LL\_FMPI2C\_GetMode
- CR1 SMBDEN LL\_FMPI2C\_GetMode

## LL\_FMPI2C\_EnableSMBusAlert

### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableSMBusAlert (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Enable SMBus alert (Host or Device mode)

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- SMBus Device mode: SMBus Alert pin is driven 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\_FMPI2C\_EnableSMBusAlert

## LL\_FMPI2C\_DisableSMBusAlert

### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableSMBusAlert (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Disable SMBus alert (Host or Device mode)

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not driven (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\_FMPI2C\_DisableSMBusAlert

**LL\_FMPI2C\_IsEnabledSMBusAlert**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledSMBusAlert (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if SMBus alert (Host or Device mode) is enabled or disabled.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Notes**

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

**Reference Manual to LL API cross reference:**

- CR1 ALERTEN LL\_FMPI2C\_IsEnabledSMBusAlert

**LL\_FMPI2C\_EnableSMBusPEC**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableSMBusPEC (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable SMBus Packet Error Calculation (PEC).

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

**Reference Manual to LL API cross reference:**

- CR1 PECEN LL\_FMPI2C\_EnableSMBusPEC

### LL\_FMPI2C\_DisableSMBusPEC

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableSMBusPEC (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable SMBus Packet Error Calculation (PEC).

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Notes

- Macro IS\_FMP2SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

#### Reference Manual to LL API cross reference:

- CR1 PECEN LL\_FMPI2C\_DisableSMBusPEC

### LL\_FMPI2C\_IsEnabledSMBusPEC

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledSMBusPEC (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Notes

- Macro IS\_FMP2SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

#### Reference Manual to LL API cross reference:

- CR1 PECEN LL\_FMPI2C\_IsEnabledSMBusPEC

### LL\_FMPI2C\_ConfigSMBusTimeout

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_ConfigSMBusTimeout (FMPI2C_TypeDef * FMPI2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)
```

#### Function description

Configure the SMBus Clock Timeout.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **TimeoutA:** This parameter must be a value between Min\_Data=0 and Max\_Data=0xFFFF.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SCL\_LOW
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SDA\_SCL\_HIGH
- **TimeoutB:**

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/or TimeoutB).

### Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL\_FMPI2C\_ConfigSMBusTimeout
- TIMEOUTR TIDLE LL\_FMPI2C\_ConfigSMBusTimeout
- TIMEOUTR TIMEOUTB LL\_FMPI2C\_ConfigSMBusTimeout

#### LL\_FMPI2C\_SetSMBusTimeoutA

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetSMBusTimeoutA (FMPI2C_TypeDef * FMPI2Cx, uint32_t TimeoutA)
```

### Function description

Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **TimeoutA:** This parameter must be a value between Min\_Data=0 and Max\_Data=0xFFFF.

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- These bits can only be programmed when TimeoutA is disabled.

### Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL\_FMPI2C\_SetSMBusTimeoutA

#### LL\_FMPI2C\_GetSMBusTimeoutA

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetSMBusTimeoutA (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the SMBus Clock TimeoutA setting.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.



### Return values

- **Value:** between Min\_Data=0 and Max\_Data=0xFF

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL\_FMPI2C\_GetSMBusTimeoutA

### LL\_FMPI2C\_SetSMBusTimeoutAMode

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetSMBusTimeoutAMode (FMPI2C_TypeDef * FMPI2Cx, uint32_t TimeoutAMode)
```

### Function description

Set the SMBus Clock TimeoutA mode.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SCL\_LOW
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SDA\_SCL\_HIGH

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- This bit can only be programmed when TimeoutA is disabled.

### Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL\_FMPI2C\_SetSMBusTimeoutAMode

### LL\_FMPI2C\_GetSMBusTimeoutAMode

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetSMBusTimeoutAMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the SMBus Clock TimeoutA mode.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SCL\_LOW
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SDA\_SCL\_HIGH

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

#### Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL\_FMPI2C\_GetSMBusTimeoutAMode

#### LL\_FMPI2C\_SetSMBusTimeoutB

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetSMBusTimeoutB (FMPI2C_TypeDef * FMPI2Cx, uint32_t TimeoutB)
```

#### Function description

Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **TimeoutB:** This parameter must be a value between Min\_Data=0 and Max\_Data=0xFFFF.

#### Return values

- **None:**

#### Notes

- Macro IS\_FMP2SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- These bits can only be programmed when TimeoutB is disabled.

#### Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL\_FMPI2C\_SetSMBusTimeoutB

#### LL\_FMPI2C\_GetSMBusTimeoutB

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetSMBusTimeoutB (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Get the SMBus Extended Cumulative Clock TimeoutB setting.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **Value:** between Min\_Data=0 and Max\_Data=0xFFFF

#### Notes

- Macro IS\_FMP2SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

#### Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL\_FMPI2C\_GetSMBusTimeoutB

#### LL\_FMPI2C\_EnableSMBusTimeout

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableSMBusTimeout (FMPI2C_TypeDef * FMPI2Cx, uint32_t ClockTimeout)
```

#### Function description

Enable the SMBus Clock Timeout.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA
  - LL\_FMPI2C\_SMBUS\_TIMEOUTB
  - LL\_FMPI2C\_SMBUS\_ALL\_TIMEOUT

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL\_FMPI2C\_EnableSMBusTimeout
- TIMEOUTR TEXTEN LL\_FMPI2C\_EnableSMBusTimeout

#### **LL\_FMPI2C\_DisableSMBusTimeout**

### Function name

**\_\_STATIC\_INLINE void LL\_FMPI2C\_DisableSMBusTimeout (FMPI2C\_TypeDef \* FMPI2Cx, uint32\_t ClockTimeout)**

### Function description

Disable the SMBus Clock Timeout.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA
  - LL\_FMPI2C\_SMBUS\_TIMEOUTB
  - LL\_FMPI2C\_SMBUS\_ALL\_TIMEOUT

### Return values

- **None:**

### Notes

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL\_FMPI2C\_DisableSMBusTimeout
- TIMEOUTR TEXTEN LL\_FMPI2C\_DisableSMBusTimeout

#### **LL\_FMPI2C\_IsEnabledSMBusTimeout**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_FMPI2C\_IsEnabledSMBusTimeout (FMPI2C\_TypeDef \* FMPI2Cx, uint32\_t ClockTimeout)**

### Function description

Check if the SMBus Clock Timeout is enabled or disabled.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL\_FMPI2C\_SMBUS\_TIMEOUTA
  - LL\_FMPI2C\_SMBUS\_TIMEOUTB
  - LL\_FMPI2C\_SMBUS\_ALL\_TIMEOUT

**Return values**

- **State:** of bit (1 or 0).

**Notes**

- Macro IS\_FMPMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

**Reference Manual to LL API cross reference:**

- TIMEOUTR TIMOUTEN LL\_FMPI2C\_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL\_FMPI2C\_IsEnabledSMBusTimeout

**LL\_FMPI2C\_EnableIT\_TX**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_TX (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable TXIS interrupt.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 TXIE LL\_FMPI2C\_EnableIT\_TX

**LL\_FMPI2C\_DisableIT\_TX**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_TX (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable TXIS interrupt.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 TXIE LL\_FMPI2C\_DisableIT\_TX

**LL\_FMPI2C\_IsEnabledIT\_TX**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_TX (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Check if the TXIS Interrupt is enabled or disabled.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **State**: of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR1 TXIE LL\_FMPI2C\_IsEnabledIT\_TX

### LL\_FMPI2C\_EnableIT\_RX

### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_RX (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Enable RXNE interrupt.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- CR1 RXIE LL\_FMPI2C\_EnableIT\_RX

### LL\_FMPI2C\_DisableIT\_RX

### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_RX (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Disable RXNE interrupt.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- CR1 RXIE LL\_FMPI2C\_DisableIT\_RX

### LL\_FMPI2C\_IsEnabledIT\_RX

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_RX (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Check if the RXNE Interrupt is enabled or disabled.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR1 RXIE LL\_FMPI2C\_IsEnabledIT\_RX

**LL\_FMPI2C\_EnableIT\_ADDR**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_ADDR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable Address match interrupt (slave mode only).

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 ADDRIE LL\_FMPI2C\_EnableIT\_ADDR

**LL\_FMPI2C\_DisableIT\_ADDR**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_ADDR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable Address match interrupt (slave mode only).

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 ADDRIE LL\_FMPI2C\_DisableIT\_ADDR

**LL\_FMPI2C\_IsEnabledIT\_ADDR**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_ADDR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if Address match interrupt is enabled or disabled.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR1 ADDRIE LL\_FMPI2C\_IsEnabledIT\_ADDR

### LL\_FMPI2C\_EnableIT\_NACK

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_NACK (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable Not acknowledge received interrupt.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 NACKIE LL\_FMPI2C\_EnableIT\_NACK

### LL\_FMPI2C\_DisableIT\_NACK

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_NACK (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable Not acknowledge received interrupt.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 NACKIE LL\_FMPI2C\_DisableIT\_NACK

### LL\_FMPI2C\_IsEnabledIT\_NACK

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_NACK (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if Not acknowledge received interrupt is enabled or disabled.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR1 NACKIE LL\_FMPI2C\_IsEnabledIT\_NACK

### LL\_FMPI2C\_EnableIT\_STOP

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_STOP (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable STOP detection interrupt.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **None**:

**Reference Manual to LL API cross reference:**

- CR1 STOPIE LL\_FMPI2C\_EnableIT\_STOP

**LL\_FMPI2C\_DisableIT\_STOP**

**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_STOP (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable STOP detection interrupt.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **None**:

**Reference Manual to LL API cross reference:**

- CR1 STOPIE LL\_FMPI2C\_DisableIT\_STOP

**LL\_FMPI2C\_IsEnabledIT\_STOP**

**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_STOP (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if STOP detection interrupt is enabled or disabled.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **State**: of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR1 STOPIE LL\_FMPI2C\_IsEnabledIT\_STOP

**LL\_FMPI2C\_EnableIT\_TC**

**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_TC (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable Transfer Complete interrupt.

**Parameters**

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_EnableIT\_TC

**LL\_FMPI2C\_DisableIT\_TC**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_TC (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable Transfer Complete interrupt.

**Parameters**

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_DisableIT\_TC

**LL\_FMPI2C\_IsEnabledIT\_TC**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_TC (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if Transfer Complete interrupt is enabled or disabled.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **State**: of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR1 TCIE LL\_FMPI2C\_IsEnabledIT\_TC

**LL\_FMPI2C\_EnableIT\_ERR**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableIT_ERR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable Error interrupts.

**Parameters**

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro IS\_FMP2C\_SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx 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\_FMPI2C\_EnableIT\_ERR

**LL\_FMPI2C\_DisableIT\_ERR**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableIT_ERR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable Error interrupts.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro IS\_FMP2C\_SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx 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\_FMPI2C\_DisableIT\_ERR

**LL\_FMPI2C\_IsEnabledIT\_ERR**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledIT_ERR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Check if Error interrupts are enabled or disabled.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR1 ERRIE LL\_FMPI2C\_IsEnabledIT\_ERR

**LL\_FMPI2C\_IsActiveFlag\_TXE**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_TXE (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Indicate the status of Transmit data register empty flag.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_TXE

#### LL\_FMPI2C\_IsActiveFlag\_TXIS

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_TXIS (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Indicate the status of Transmit interrupt flag.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_TXIS

#### LL\_FMPI2C\_IsActiveFlag\_RXNE

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_RXNE (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Indicate the status of Receive data register not empty flag.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_RXNE

#### LL\_FMPI2C\_IsActiveFlag\_ADDR

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_ADDR (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Indicate the status of Address matched flag (slave mode).

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_ADDR

**LL\_FMPI2C\_IsActiveFlag\_NACK**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_FMPI2C\_IsActiveFlag\_NACK (FMPI2C\_TypeDef \* FMPI2Cx)**

### Function description

Indicate the status of Not Acknowledge received flag.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_NACK

**LL\_FMPI2C\_IsActiveFlag\_STOP**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_FMPI2C\_IsActiveFlag\_STOP (FMPI2C\_TypeDef \* FMPI2Cx)**

### Function description

Indicate the status of Stop detection flag.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_STOP

### LL\_FMPI2C\_IsActiveFlag\_TC

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_TC (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Indicate the status of Transfer complete flag (master mode).

#### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_TC

### LL\_FMPI2C\_IsActiveFlag\_TCR

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_TCR (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Indicate the status of Transfer complete flag (master mode).

#### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_TCR

### LL\_FMPI2C\_IsActiveFlag\_BERR

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_BERR (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Indicate the status of Bus error flag.

#### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_BERR

**LL\_FMPI2C\_IsActiveFlag\_ARLO**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_ARLO (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Indicate the status of Arbitration lost flag.

**Parameters**

- **FMPI2Cx:** FMPI2C 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\_FMPI2C\_IsActiveFlag\_ARLO

**LL\_FMPI2C\_IsActiveFlag\_OVR**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_OVR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Indicate the status of Overrun/Underrun flag (slave mode).

**Parameters**

- **FMPI2Cx:** FMPI2C 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\_FMPI2C\_IsActiveFlag\_OVR

**LL\_FMPI2C\_IsActiveSMBusFlag\_PECERR**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveSMBusFlag_PECERR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Indicate the status of SMBus PEC error flag in reception.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Notes**

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx 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_FMPI2C_IsActiveSMBusFlag_PECERR`

**LL\_FMPI2C\_IsActiveSMBusFlag\_TIMEOUT**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveSMBusFlag_TIMEOUT (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Indicate the status of SMBus Timeout detection flag.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Notes**

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.
- **RESET:** Clear default value. **SET:** When a timeout or extended clock timeout occurs.

**Reference Manual to LL API cross reference:**

- `ISR_TIMEOUT_LL_FMPI2C_IsActiveSMBusFlag_TIMEOUT`

**LL\_FMPI2C\_IsActiveSMBusFlag\_ALERT**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveSMBusFlag_ALERT (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Indicate the status of SMBus alert flag.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Notes**

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx 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_FMPI2C_IsActiveSMBusFlag_ALERT`

**LL\_FMPI2C\_IsActiveFlag\_BUSY**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsActiveFlag_BUSY (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Indicate the status of Bus Busy flag.

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_IsActiveFlag\_BUSY

### LL\_FMPI2C\_ClearFlag\_ADDR

### Function name

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_ADDR (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Clear Address Matched flag.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- ICR ADDR CF LL\_FMPI2C\_ClearFlag\_ADDR

### LL\_FMPI2C\_ClearFlag\_NACK

### Function name

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_NACK (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Clear Not Acknowledge flag.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- ICR NACK CF LL\_FMPI2C\_ClearFlag\_NACK

### LL\_FMPI2C\_ClearFlag\_STOP

### Function name

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_STOP (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Clear Stop detection flag.



#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- ICR STOPCF LL\_FMPI2C\_ClearFlag\_STOP

**LL\_FMPI2C\_ClearFlag\_TXE**

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_TXE (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Clear Transmit data register empty flag (TXE).

#### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_ClearFlag\_TXE

**LL\_FMPI2C\_ClearFlag\_BERR**

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_BERR (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Clear Bus error flag.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- ICR BERRCF LL\_FMPI2C\_ClearFlag\_BERR

**LL\_FMPI2C\_ClearFlag\_ARLO**

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_ARLO (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Clear Arbitration lost flag.

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- ICR ARLOCF LL\_FMPI2C\_ClearFlag\_ARLO

**LL\_FMPI2C\_ClearFlag\_OVR**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_ClearFlag_OVR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Clear Overrun/Underrun flag.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- ICR OVRFCF LL\_FMPI2C\_ClearFlag\_OVR

**LL\_FMPI2C\_ClearSMBusFlag\_PECERR**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_ClearSMBusFlag_PECERR (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Clear SMBus PEC error flag.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro IS\_FMPUSBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

**Reference Manual to LL API cross reference:**

- ICR PECCF LL\_FMPI2C\_ClearSMBusFlag\_PECERR

**LL\_FMPI2C\_ClearSMBusFlag\_TIMEOUT**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_ClearSMBusFlag_TIMEOUT (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Clear SMBus Timeout detection flag.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

**Reference Manual to LL API cross reference:**

- ICR TIMEOUTCF `LL_FMPI2C_ClearSMBusFlag_TIMEOUT`

**LL\_FMPI2C\_ClearSMBusFlag\_ALERT**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_ClearSMBusFlag_ALERT (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Clear SMBus Alert flag.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

**Reference Manual to LL API cross reference:**

- ICR ALERTCF `LL_FMPI2C_ClearSMBusFlag_ALERT`

**LL\_FMPI2C\_EnableAutoEndMode**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_EnableAutoEndMode (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Enable automatic STOP condition generation (master mode).

**Parameters**

- **FMPI2Cx:** FMPI2C 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_FMPI2C_EnableAutoEndMode`

**LL\_FMPI2C\_DisableAutoEndMode**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_DisableAutoEndMode (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Disable automatic STOP condition generation (master mode).

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_DisableAutoEndMode

#### LL\_FMPI2C\_IsEnabledAutoEndMode

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledAutoEndMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Check if automatic STOP condition is enabled or disabled.

### Parameters

- **FMPI2Cx**: FMPI2C Instance.

### Return values

- **State**: of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR2 AUTOEND LL\_FMPI2C\_IsEnabledAutoEndMode

#### LL\_FMPI2C\_EnableReloadMode

### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableReloadMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Enable reload mode (master mode).

### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_EnableReloadMode

#### LL\_FMPI2C\_DisableReloadMode

### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableReloadMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Disable reload mode (master mode).

### Parameters

- **FMPI2Cx:** FMPI2C 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\_FMPI2C\_DisableReloadMode

#### LL\_FMPI2C\_IsEnabledReloadMode

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledReloadMode (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Check if reload mode is enabled or disabled.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR2 RELOAD LL\_FMPI2C\_IsEnabledReloadMode

#### LL\_FMPI2C\_SetTransferSize

### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetTransferSize (FMPI2C_TypeDef * FMPI2Cx, uint32_t TransferSize)
```

### Function description

Configure the number of bytes for transfer.

### Parameters

- **FMPI2Cx:** FMPI2C 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\_FMPI2C\_SetTransferSize

#### LL\_FMPI2C\_GetTransferSize

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetTransferSize (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the number of bytes configured for transfer.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **Value:** between Min\_Data=0x0 and Max\_Data=0xFF

**Reference Manual to LL API cross reference:**

- CR2 NBYTES LL\_FMPI2C\_GetTransferSize

**LL\_FMPI2C\_AcknowledgeNextData**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_AcknowledgeNextData (FMPI2C_TypeDef * FMPI2Cx, 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**

- **FMPI2Cx:** FMPI2C Instance.
- **TypeAcknowledge:** This parameter can be one of the following values:
  - LL\_FMPI2C\_ACK
  - LL\_FMPI2C\_NACK

**Return values**

- **None:**

**Notes**

- Usage in Slave mode only.

**Reference Manual to LL API cross reference:**

- CR2 NACK LL\_FMPI2C\_AcknowledgeNextData

**LL\_FMPI2C\_GenerateStartCondition**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_GenerateStartCondition (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Generate a START or RESTART condition.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **None:**

**Notes**

- The START bit can be set even if bus is BUSY or FMPI2C is in slave mode. This action has no effect when RELOAD is set.

**Reference Manual to LL API cross reference:**

- CR2 START LL\_FMPI2C\_GenerateStartCondition

### LL\_FMPI2C\_GenerateStopCondition

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_GenerateStopCondition (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Generate a STOP condition after the current byte transfer (master mode).

#### Parameters

- **FMPI2Cx**: FMPI2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR2 STOP LL\_FMPI2C\_GenerateStopCondition

### LL\_FMPI2C\_EnableAuto10BitRead

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_EnableAuto10BitRead (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Enable automatic RESTART Read request condition for 10bit address header (master mode).

#### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_EnableAuto10BitRead

### LL\_FMPI2C\_DisableAuto10BitRead

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_DisableAuto10BitRead (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Disable automatic RESTART Read request condition for 10bit address header (master mode).

#### Parameters

- **FMPI2Cx**: FMPI2C 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\_FMPI2C\_DisableAuto10BitRead

### LL\_FMPI2C\_IsEnabledAuto10BitRead

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_IsEnabledAuto10BitRead (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR2 HEAD10R LL\_FMPI2C\_IsEnabledAuto10BitRead

### LL\_FMPI2C\_SetTransferRequest

#### Function name

```
__STATIC_INLINE void LL_FMPI2C_SetTransferRequest (FMPI2C_TypeDef * FMPI2Cx, uint32_t TransferRequest)
```

#### Function description

Configure the transfer direction (master mode).

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **TransferRequest:** This parameter can be one of the following values:
  - LL\_FMPI2C\_REQUEST\_WRITE
  - LL\_FMPI2C\_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\_FMPI2C\_SetTransferRequest

### LL\_FMPI2C\_GetTransferRequest

#### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetTransferRequest (FMPI2C_TypeDef * FMPI2Cx)
```

#### Function description

Get the transfer direction requested (master mode).

#### Parameters

- **FMPI2Cx:** FMPI2C Instance.

#### Return values

- **Returned:** value can be one of the following values:
  - LL\_FMPI2C\_REQUEST\_WRITE
  - LL\_FMPI2C\_REQUEST\_READ



**Reference Manual to LL API cross reference:**

- CR2 RD\_WRN LL\_FMPI2C\_GetTransferRequest

**LL\_FMPI2C\_SetSlaveAddr**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_SetSlaveAddr (FMPI2C_TypeDef * FMPI2Cx, uint32_t SlaveAddr)
```

**Function description**

Configure the slave address for transfer (master mode).

**Parameters**

- **FMPI2Cx:** FMPI2C 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\_FMPI2C\_SetSlaveAddr

**LL\_FMPI2C\_GetSlaveAddr**
**Function name**

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetSlaveAddr (FMPI2C_TypeDef * FMPI2Cx)
```

**Function description**

Get the slave address programmed for transfer.

**Parameters**

- **FMPI2Cx:** FMPI2C Instance.

**Return values**

- **Value:** between Min\_Data=0x0 and Max\_Data=0x3F

**Reference Manual to LL API cross reference:**

- CR2 SADD LL\_FMPI2C\_GetSlaveAddr

**LL\_FMPI2C\_HandleTransfer**
**Function name**

```
__STATIC_INLINE void LL_FMPI2C_HandleTransfer (FMPI2C_TypeDef * FMPI2Cx, uint32_t SlaveAddr,
uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)
```

**Function description**

Handles FMPI2Cx communication when starting transfer or during transfer (TC or TCR flag are set).

## Parameters

- **FMPI2Cx:** FMPI2C Instance.
- **SlaveAddr:** Specifies the slave address to be programmed.
- **SlaveAddrSize:** This parameter can be one of the following values:
  - LL\_FMPI2C\_ADDRSLAVE\_7BIT
  - LL\_FMPI2C\_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\_FMPI2C\_MODE\_RELOAD
  - LL\_FMPI2C\_MODE\_AUTOEND
  - LL\_FMPI2C\_MODE\_SOFTEND
  - LL\_FMPI2C\_MODE\_SMBUS\_RELOAD
  - LL\_FMPI2C\_MODE\_SMBUS\_AUTOEND\_NO\_PEC
  - LL\_FMPI2C\_MODE\_SMBUS\_SOFTEND\_NO\_PEC
  - LL\_FMPI2C\_MODE\_SMBUS\_AUTOEND\_WITH\_PEC
  - LL\_FMPI2C\_MODE\_SMBUS\_SOFTEND\_WITH\_PEC
- **Request:** This parameter can be one of the following values:
  - LL\_FMPI2C\_GENERATE\_NOSTARTSTOP
  - LL\_FMPI2C\_GENERATE\_STOP
  - LL\_FMPI2C\_GENERATE\_START\_READ
  - LL\_FMPI2C\_GENERATE\_START\_WRITE
  - LL\_FMPI2C\_GENERATE\_RESTART\_7BIT\_READ
  - LL\_FMPI2C\_GENERATE\_RESTART\_7BIT\_WRITE
  - LL\_FMPI2C\_GENERATE\_RESTART\_10BIT\_READ
  - LL\_FMPI2C\_GENERATE\_RESTART\_10BIT\_WRITE

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- CR2 SADD LL\_FMPI2C\_HandleTransfer
- CR2 ADD10 LL\_FMPI2C\_HandleTransfer
- CR2 RD\_WRN LL\_FMPI2C\_HandleTransfer
- CR2 START LL\_FMPI2C\_HandleTransfer
- CR2 STOP LL\_FMPI2C\_HandleTransfer
- CR2 RELOAD LL\_FMPI2C\_HandleTransfer
- CR2 NBYTES LL\_FMPI2C\_HandleTransfer
- CR2 AUTOEND LL\_FMPI2C\_HandleTransfer
- CR2 HEAD10R LL\_FMPI2C\_HandleTransfer

## LL\_FMPI2C\_GetTransferDirection

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetTransferDirection (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Indicate the value of transfer direction (slave mode).

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_FMPI2C\_DIRECTION\_WRITE
  - LL\_FMPI2C\_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\_FMPI2C\_GetTransferDirection

#### LL\_FMPI2C\_GetAddressMatchCode

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_FMPI2C\_GetAddressMatchCode (FMPI2C\_TypeDef \* FMPI2Cx)**

### Function description

Return the slave matched address.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Value:** between Min\_Data=0x00 and Max\_Data=0x3F

### Reference Manual to LL API cross reference:

- ISR ADDCODE LL\_FMPI2C\_GetAddressMatchCode

#### LL\_FMPI2C\_EnableSMBusPECCompare

### Function name

**\_\_STATIC\_INLINE void LL\_FMPI2C\_EnableSMBusPECCompare (FMPI2C\_TypeDef \* FMPI2Cx)**

### Function description

Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **None:**

### Notes

- Macro IS\_FMP2C\_SMBUS\_ALL\_INSTANCE(FMPI2Cx) can be used to check whether or not SMBus feature is supported by the FMPI2Cx 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\_FMPI2C\_EnableSMBusPECCompare

#### LL\_FMPI2C\_IsEnabledSMBusPECCompare

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_FMPI2C\_IsEnabledSMBusPECCompare (FMPI2C\_TypeDef \* FMPI2Cx)**

### Function description

Check if the SMBus Packet Error byte internal comparison is requested or not.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **State:** of bit (1 or 0).

### Notes

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- CR2 PECBYTE LL\_FMPI2C\_IsEnabledSMBusPECCompare

#### LL\_FMPI2C\_GetSMBusPEC

### Function name

```
__STATIC_INLINE uint32_t LL_FMPI2C_GetSMBusPEC (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Get the SMBus Packet Error byte calculated.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Value:** between `Min_Data=0x00` and `Max_Data=0xFF`

### Notes

- Macro `IS_FMPMBUS_ALL_INSTANCE(FMPI2Cx)` can be used to check whether or not SMBus feature is supported by the FMPI2Cx Instance.

### Reference Manual to LL API cross reference:

- PECCR PEC LL\_FMPI2C\_GetSMBusPEC

#### LL\_FMPI2C\_ReceiveData8

### Function name

```
__STATIC_INLINE uint8_t LL_FMPI2C_ReceiveData8 (FMPI2C_TypeDef * FMPI2Cx)
```

### Function description

Read Receive Data register.

### Parameters

- **FMPI2Cx:** FMPI2C Instance.

### Return values

- **Value:** between `Min_Data=0x00` and `Max_Data=0xFF`

### Reference Manual to LL API cross reference:

- RXDR RXDATA LL\_FMPI2C\_ReceiveData8

#### LL\_FMPI2C\_TransmitData8

### Function name

```
__STATIC_INLINE void LL_FMPI2C_TransmitData8 (FMPI2C_TypeDef * FMPI2Cx, uint8_t Data)
```

### Function description

Write in Transmit Data Register .

### Parameters

- **FMPI2Cx**: FMPI2C Instance.
- **Data**: Value between Min\_Data=0x00 and Max\_Data=0xFF

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- TXDR TXDATA LL\_FMPI2C\_TransmitData8

#### LL\_FMPI2C\_Init

### Function name

ErrorStatus LL\_FMPI2C\_Init (FMPI2C\_TypeDef \* FMPI2Cx, LL\_FMPI2C\_InitTypeDef \* FMPI2C\_InitStruct)

### Function description

#### LL\_FMPI2C\_DeInit

### Function name

ErrorStatus LL\_FMPI2C\_DeInit (FMPI2C\_TypeDef \* FMPI2Cx)

### Function description

#### LL\_FMPI2C\_StructInit

### Function name

void LL\_FMPI2C\_StructInit (LL\_FMPI2C\_InitTypeDef \* FMPI2C\_InitStruct)

### Function description

## 80.3 FMPI2C Firmware driver defines

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

### 80.3.1 FMPI2C

FMPI2C

#### **Master Addressing Mode**

#### LL\_FMPI2C\_ADDRESSING\_MODE\_7BIT

Master operates in 7-bit addressing mode.

#### LL\_FMPI2C\_ADDRESSING\_MODE\_10BIT

Master operates in 10-bit addressing mode.

#### **Slave Address Length**

#### LL\_FMPI2C\_ADDRSLAVE\_7BIT

Slave Address in 7-bit.

#### LL\_FMPI2C\_ADDRSLAVE\_10BIT

Slave Address in 10-bit.

#### **Analog Filter Selection**

**LL\_FMPI2C\_ANALOGFILTER\_ENABLE**

Analog filter is enabled.

**LL\_FMPI2C\_ANALOGFILTER\_DISABLE**

Analog filter is disabled.

**Clear Flags Defines**

**LL\_FMPI2C\_ICR\_ADDRCF**

Address Matched flag

**LL\_FMPI2C\_ICR\_NACKCF**

Not Acknowledge flag

**LL\_FMPI2C\_ICR\_STOPCF**

Stop detection flag

**LL\_FMPI2C\_ICR\_BERRCF**

Bus error flag

**LL\_FMPI2C\_ICR\_ARLOCF**

Arbitration Lost flag

**LL\_FMPI2C\_ICR\_OVRCF**

Overrun/Underrun flag

**LL\_FMPI2C\_ICR\_PECCF**

PEC error flag

**LL\_FMPI2C\_ICR\_TIMEOUTCF**

Timeout detection flag

**LL\_FMPI2C\_ICR\_ALERTCF**

Alert flag

**Read Write Direction**

**LL\_FMPI2C\_DIRECTION\_WRITE**

Write transfer request by master, slave enters receiver mode.

**LL\_FMPI2C\_DIRECTION\_READ**

Read transfer request by master, slave enters transmitter mode.

**DMA Register Data**

**LL\_FMPI2C\_DMA\_REG\_DATA\_TRANSMIT**

Get address of data register used for transmission

**LL\_FMPI2C\_DMA\_REG\_DATA\_RECEIVE**

Get address of data register used for reception

**Start And Stop Generation**

**LL\_FMPI2C\_GENERATE\_NOSTARTSTOP**

Don't Generate Stop and Start condition.

**LL\_FMPI2C\_GENERATE\_STOP**

Generate Stop condition (Size should be set to 0).

**LL\_FMPI2C\_GENERATE\_START\_READ**

Generate Start for read request.

**LL\_FMPI2C\_GENERATE\_START\_WRITE**

Generate Start for write request.

**LL\_FMPI2C\_GENERATE\_RESTART\_7BIT\_READ**

Generate Restart for read request, slave 7Bit address.

**LL\_FMPI2C\_GENERATE\_RESTART\_7BIT\_WRITE**

Generate Restart for write request, slave 7Bit address.

**LL\_FMPI2C\_GENERATE\_RESTART\_10BIT\_READ**

Generate Restart for read request, slave 10Bit address.

**LL\_FMPI2C\_GENERATE\_RESTART\_10BIT\_WRITE**

Generate Restart for write request, slave 10Bit address.

***Get Flags Defines***

**LL\_FMPI2C\_ISR\_TXE**

Transmit data register empty

**LL\_FMPI2C\_ISR\_TXIS**

Transmit interrupt status

**LL\_FMPI2C\_ISR\_RXNE**

Receive data register not empty

**LL\_FMPI2C\_ISR\_ADDR**

Address matched (slave mode)

**LL\_FMPI2C\_ISR\_NACKF**

Not Acknowledge received flag

**LL\_FMPI2C\_ISR\_STOPF**

Stop detection flag

**LL\_FMPI2C\_ISR\_TC**

Transfer Complete (master mode)

**LL\_FMPI2C\_ISR\_TCR**

Transfer Complete Reload

**LL\_FMPI2C\_ISR\_BERR**

Bus error

**LL\_FMPI2C\_ISR\_ARLO**

Arbitration lost

**LL\_FMPI2C\_ISR\_OVR**

Overrun/Underrun (slave mode)

**LL\_FMPI2C\_ISR\_PECERR**

PEC Error in reception (SMBus mode)

**LL\_FMPI2C\_ISR\_TIMEOUT**

Timeout detection flag (SMBus mode)

**LL\_FMPI2C\_ISR\_ALERT**

SMBus alert (SMBus mode)

**LL\_FMPI2C\_ISR\_BUSY**

Bus busy

**Acknowledge Generation**

**LL\_FMPI2C\_ACK**

ACK is sent after current received byte.

**LL\_FMPI2C\_NACK**

NACK is sent after current received byte.

**IT Defines**

**LL\_FMPI2C\_CR1\_TXIE**

TX Interrupt enable

**LL\_FMPI2C\_CR1\_RXIE**

RX Interrupt enable

**LL\_FMPI2C\_CR1\_ADDRIE**

Address match Interrupt enable (slave only)

**LL\_FMPI2C\_CR1\_NACKIE**

Not acknowledge received Interrupt enable

**LL\_FMPI2C\_CR1\_STOPIE**

STOP detection Interrupt enable

**LL\_FMPI2C\_CR1\_TCIE**

Transfer Complete interrupt enable

**LL\_FMPI2C\_CR1\_ERRIE**

Error interrupts enable

**Transfer End Mode**

**LL\_FMPI2C\_MODE\_RELOAD**

Enable FMPI2C Reload mode.

**LL\_FMPI2C\_MODE\_AUTOEND**

Enable FMPI2C Automatic end mode with no HW PEC comparison.

**LL\_FMPI2C\_MODE\_SOFTEND**

Enable FMPI2C Software end mode with no HW PEC comparison.

**LL\_FMPI2C\_MODE\_SMBUS\_RELOAD**

Enable SMBUS Automatic end mode with HW PEC comparison.

**LL\_FMPI2C\_MODE\_SMBUS\_AUTOEND\_NO\_PEC**

Enable SMBUS Automatic end mode with HW PEC comparison.

**LL\_FMPI2C\_MODE\_SMBUS\_SOFTEND\_NO\_PEC**

Enable SMBUS Software end mode with HW PEC comparison.

**LL\_FMPI2C\_MODE\_SMBUS\_AUTOEND\_WITH\_PEC**

Enable SMBUS Automatic end mode with HW PEC comparison.

**LL\_FMPI2C\_MODE\_SMBUS\_SOFTEND\_WITH\_PEC**

Enable SMBUS Software end mode with HW PEC comparison.

**Own Address 1 Length**



**LL\_FMPI2C\_OWNADDRESS1\_7BIT**

Own address 1 is a 7-bit address.

**LL\_FMPI2C\_OWNADDRESS1\_10BIT**

Own address 1 is a 10-bit address.

**Own Address 2 Masks**

**LL\_FMPI2C\_OWNADDRESS2\_NOMASK**

Own Address2 No mask.

**LL\_FMPI2C\_OWNADDRESS2\_MASK01**

Only Address2 bits[7:2] are compared.

**LL\_FMPI2C\_OWNADDRESS2\_MASK02**

Only Address2 bits[7:3] are compared.

**LL\_FMPI2C\_OWNADDRESS2\_MASK03**

Only Address2 bits[7:4] are compared.

**LL\_FMPI2C\_OWNADDRESS2\_MASK04**

Only Address2 bits[7:5] are compared.

**LL\_FMPI2C\_OWNADDRESS2\_MASK05**

Only Address2 bits[7:6] are compared.

**LL\_FMPI2C\_OWNADDRESS2\_MASK06**

Only Address2 bits[7] are compared.

**LL\_FMPI2C\_OWNADDRESS2\_MASK07**

No comparison is done. All Address2 are acknowledged.

**Peripheral Mode**

**LL\_FMPI2C\_MODE\_I2C**

FMPI2C Master or Slave mode

**LL\_FMPI2C\_MODE\_SMBUS\_HOST**

SMBus Host address acknowledge

**LL\_FMPI2C\_MODE\_SMBUS\_DEVICE**

SMBus Device default mode (Default address not acknowledge)

**LL\_FMPI2C\_MODE\_SMBUS\_DEVICE\_ARP**

SMBus Device Default address acknowledge

**Transfer Request Direction**

**LL\_FMPI2C\_REQUEST\_WRITE**

Master request a write transfer.

**LL\_FMPI2C\_REQUEST\_READ**

Master request a read transfer.

**SMBus TimeoutA Mode SCL SDA Timeout**

**LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SCL\_LOW**

TimeoutA is used to detect SCL low level timeout.

**LL\_FMPI2C\_SMBUS\_TIMEOUTA\_MODE\_SDA\_SCL\_HIGH**

TimeoutA is used to detect both SCL and SDA high level timeout.

**SMBus Timeout Selection**

**LL\_FMPI2C\_SMBUS\_TIMEOUTA**

TimeoutA enable bit

**LL\_FMPI2C\_SMBUS\_TIMEOUTB**

TimeoutB (extended clock) enable bit

**LL\_FMPI2C\_SMBUS\_ALL\_TIMEOUT**

TimeoutA and TimeoutB (extended clock) enable bits

**Convert SDA SCL timings**

**\_\_LL\_FMPI2C\_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\_FMPI2C\_WriteReg**

**Description:**

- Write a value in FMPI2C register.

**Parameters:**

- **\_\_INSTANCE\_\_**: FMPI2C Instance
- **\_\_REG\_\_**: Register to be written
- **\_\_VALUE\_\_**: Value to be written in the register

**Return value:**

- None

**LL\_FMPI2C\_ReadReg**

**Description:**

- Read a value in FMPI2C register.

**Parameters:**

- **\_\_INSTANCE\_\_**: FMPI2C Instance
- **\_\_REG\_\_**: Register to be read

**Return value:**

- Register: value

## 81 LL EXTI Generic Driver

### 81.1 EXTI Firmware driver registers structures

#### 81.1.1 LL\_EXTI\_InitTypeDef

*LL\_EXTI\_InitTypeDef* is defined in the `stm32f4xx_ll_exti.h`

##### Data Fields

- *uint32\_t Line\_0\_31*
- *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](#)
- *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](#).

### 81.2 EXTI Firmware driver API description

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

#### 81.2.1 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\_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:

- IMR IMx LL\_EXTI\_EnableIT\_0\_31

### 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\_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:

- IMR IMx LL\_EXTI\_DisableIT\_0\_31

### 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\_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:

- IMR IMx LL\_EXTI\_IsEnabledIT\_0\_31

### 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\_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:

- EMR EMx LL\_EXTI\_EnableEvent\_0\_31

### 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\_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:

- EMR EMx LL\_EXTI\_DisableEvent\_0\_31

#### **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\_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:

- EMR EMx LL\_EXTI\_IsEnabledEvent\_0\_31

### 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

### 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\_RTISR 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:

- RTSR RTx LL\_EXTI\_EnableRisingTrig\_0\_31

#### **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

### 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\_RTISR 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:

- RTSR RTx LL\_EXTI\_DisableRisingTrig\_0\_31

#### **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

### 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:

- RTSR RTx LL\_EXTI\_IsEnabledRisingTrig\_0\_31

### 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

### 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:

- FTSR FTx LL\_EXTI\_EnableFallingTrig\_0\_31

#### 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

### 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:

- FTSR FTx LL\_EXTI\_DisableFallingTrig\_0\_31

#### **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

### 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:

- FTSR FTx LL\_EXTI\_IsEnabledFallingTrig\_0\_31

### 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

### Return values

- **None:**

### Notes

- If the interrupt is enabled on this line in the EXTI\_IMR, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI\_PR resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI\_PR 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:

- SWIER SWIx LL\_EXTI\_GenerateSWI\_0\_31

#### **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

### 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:

- PR PIFx LL\_EXTI\_IsActiveFlag\_0\_31

#### 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

## 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:

- PR PIFx LL\_EXTI\_ReadFlag\_0\_31

### 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

### 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:

- PR PIFx LL\_EXTI\_ClearFlag\_0\_31

### 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:
  - SUCCESS: EXTI registers are initialized
  - ERROR: not applicable

## 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:
  - SUCCESS: EXTI registers are de-initialized
  - ERROR: not applicable

## 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:**

## 81.3 EXTI Firmware driver defines

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

### 81.3.1 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\_ALL\_0\_31**

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

## 82 LL GPIO Generic Driver

### 82.1 GPIO Firmware driver registers structures

#### 82.1.1 LL\_GPIO\_InitTypeDef

*LL\_GPIO\_InitTypeDef* is defined in the `stm32f4xx_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()`.

### 82.2 GPIO Firmware driver API description

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

#### 82.2.1 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:

- BSRR 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:**

## 82.3 GPIO Firmware driver defines

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

### 82.3.1 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\_PUSH\_PULL**

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

## 83 LL I2C Generic Driver

### 83.1 I2C Firmware driver registers structures

#### 83.1.1 LL\_I2C\_InitTypeDef

*LL\_I2C\_InitTypeDef* is defined in the `stm32f4xx_ll_i2c.h`

##### Data Fields

- *uint32\_t PeripheralMode*
- *uint32\_t ClockSpeed*
- *uint32\_t DutyCycle*
- *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::ClockSpeed*  
 Specifies the clock frequency. This parameter must be set to a value lower than 400kHz (in Hz)This feature can be modified afterwards using unitary function `LL_I2C_SetClockPeriod()` or `LL_I2C_SetDutyCycle()` or `LL_I2C_SetClockSpeedMode()` or `LL_I2C_ConfigSpeed()`.
- *uint32\_t LL\_I2C\_InitTypeDef::DutyCycle*  
 Specifies the I2C fast mode duty cycle. This parameter can be a value of *I2C\_LL\_EC\_DUTYCYCLE*This feature can be modified afterwards using unitary function `LL_I2C_SetDutyCycle()`.
- *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 = 0x0F`This 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 = 0x3FF`This 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\_OWNAADDRESS1*This feature can be modified afterwards using unitary function `LL_I2C_SetOwnAddress1()`.

### 83.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

### 83.2.1 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**:

##### 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\*TPCLK1) This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will suppress the spikes with a length of up to DNF[3:0]\*TPCLK1.

### 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:

- FLTR ANOFF LL\_I2C\_ConfigFilters
- FLTR 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\*TPCLK1) This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will suppress the spikes with a length of up to DNF[3:0]\*TPCLK1.

### 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:

- FLTR 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:**

- FLTR 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:**

- FLTR ANOFF 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:**

- FLTR ANOFF 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:**

- FLTR ANOFF 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:**

- CR2 DMAEN 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:**

- CR2 DMAEN 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:**

- CR2 DMAEN 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:**

- CR2 DMAEN 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:**

- CR2 DMAEN 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:**

- CR2 DMAEN LL\_I2C\_IsEnabledDMAReq\_RX

**LL\_I2C\_DMA\_GetRegAddr**
**Function name**

```
__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx)
```

### Function description

Get the data register address used for DMA transfer.

### Parameters

- **I2Cx**: I2C Instance.

### Return values

- **Address**: of data register

### Reference Manual to LL API cross reference:

- DR DR 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\_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 ENGC 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 ENGC 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 ENGC LL\_I2C\_IsEnabledGeneralCall

### 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 ADD0 LL\_I2C\_SetOwnAddress1
- OAR1 ADD1\_7 LL\_I2C\_SetOwnAddress1
- OAR1 ADD8\_9 LL\_I2C\_SetOwnAddress1
- OAR1 ADDMODE LL\_I2C\_SetOwnAddress1

### LL\_I2C\_SetOwnAddress2

### Function name

```
__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2)
```

### Function description

Set the 7bits Own Address2.

### Parameters

- **I2Cx**: I2C Instance.
- **OwnAddress2**: This parameter must be a value between Min\_Data=0 and Max\_Data=0x7F.

### Return values

- **None**:

### Notes

- This action has no effect if own address2 is enabled.

### Reference Manual to LL API cross reference:

- OAR2 ADD2 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 ENDUAL 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 ENDUAL 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 ENDUAL LL\_I2C\_IsEnabledOwnAddress2

### LL\_I2C\_SetPeriphClock

#### Function name

```
__STATIC_INLINE void LL_I2C_SetPeriphClock (I2C_TypeDef * I2Cx, uint32_t PeriphClock)
```

#### Function description

Configure the Peripheral clock frequency.



### Parameters

- **I2Cx**: I2C Instance.
- **PeriphClock**: Peripheral Clock (in Hz)

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- CR2 FREQ LL\_I2C\_SetPeriphClock

### LL\_I2C\_GetPeriphClock

### Function name

```
__STATIC_INLINE uint32_t LL_I2C_GetPeriphClock (I2C_TypeDef * I2Cx)
```

### Function description

Get the Peripheral clock frequency.

### Parameters

- **I2Cx**: I2C Instance.

### Return values

- **Value**: of Peripheral Clock (in Hz)

### Reference Manual to LL API cross reference:

- CR2 FREQ LL\_I2C\_GetPeriphClock

### LL\_I2C\_SetDutyCycle

### Function name

```
__STATIC_INLINE void LL_I2C_SetDutyCycle (I2C_TypeDef * I2Cx, uint32_t DutyCycle)
```

### Function description

Configure the Duty cycle (Fast mode only).

### Parameters

- **I2Cx**: I2C Instance.
- **DutyCycle**: This parameter can be one of the following values:
  - LL\_I2C\_DUTYCYCLE\_2
  - LL\_I2C\_DUTYCYCLE\_16\_9

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- CCR DUTY LL\_I2C\_SetDutyCycle

### LL\_I2C\_GetDutyCycle

### Function name

```
__STATIC_INLINE uint32_t LL_I2C_GetDutyCycle (I2C_TypeDef * I2Cx)
```

### Function description

Get the Duty cycle (Fast mode only).

### Parameters

- **I2Cx**: I2C Instance.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_I2C\_DUTYCYCLE\_2
  - LL\_I2C\_DUTYCYCLE\_16\_9

### Reference Manual to LL API cross reference:

- CCR DUTY LL\_I2C\_GetDutyCycle

### LL\_I2C\_SetClockSpeedMode

### Function name

**\_\_STATIC\_INLINE void LL\_I2C\_SetClockSpeedMode (I2C\_TypeDef \* I2Cx, uint32\_t ClockSpeedMode)**

### Function description

Configure the I2C master clock speed mode.

### Parameters

- **I2Cx:** I2C Instance.
- **ClockSpeedMode:** This parameter can be one of the following values:
  - LL\_I2C\_CLOCK\_SPEED\_STANDARD\_MODE
  - LL\_I2C\_CLOCK\_SPEED\_FAST\_MODE

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CCR FS LL\_I2C\_SetClockSpeedMode

### LL\_I2C\_GetClockSpeedMode

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_I2C\_GetClockSpeedMode (I2C\_TypeDef \* I2Cx)**

### Function description

Get the the I2C master speed mode.

### Parameters

- **I2Cx:** I2C Instance.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_I2C\_CLOCK\_SPEED\_STANDARD\_MODE
  - LL\_I2C\_CLOCK\_SPEED\_FAST\_MODE

### Reference Manual to LL API cross reference:

- CCR FS LL\_I2C\_GetClockSpeedMode

### LL\_I2C\_SetRiseTime

### Function name

**\_\_STATIC\_INLINE void LL\_I2C\_SetRiseTime (I2C\_TypeDef \* I2Cx, uint32\_t RiseTime)**

### Function description

Configure the SCL, SDA rising time.

### Parameters

- **I2Cx:** I2C Instance.
- **RiseTime:** This parameter must be a value between Min\_Data=0x02 and Max\_Data=0x3F.

### Return values

- **None:**

### Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- TRISE TRISE LL\_I2C\_SetRiseTime

### LL\_I2C\_GetRiseTime

### Function name

```
__STATIC_INLINE uint32_t LL_I2C_GetRiseTime (I2C_TypeDef * I2Cx)
```

### Function description

Get the SCL, SDA rising time.

### Parameters

- **I2Cx:** I2C Instance.

### Return values

- **Value:** between Min\_Data=0x02 and Max\_Data=0x3F

### Reference Manual to LL API cross reference:

- TRISE TRISE LL\_I2C\_GetRiseTime

### LL\_I2C\_SetClockPeriod

### Function name

```
__STATIC_INLINE void LL_I2C_SetClockPeriod (I2C_TypeDef * I2Cx, uint32_t ClockPeriod)
```

### Function description

Configure the SCL high and low period.

### Parameters

- **I2Cx:** I2C Instance.
- **ClockPeriod:** This parameter must be a value between Min\_Data=0x004 and Max\_Data=0xFFFF, except in FAST DUTY mode where Min\_Data=0x001.

### Return values

- **None:**

### Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- CCR CCR LL\_I2C\_SetClockPeriod

### LL\_I2C\_GetClockPeriod

### Function name

```
__STATIC_INLINE uint32_t LL_I2C_GetClockPeriod (I2C_TypeDef * I2Cx)
```

### Function description

Get the SCL high and low period.

### Parameters

- **I2Cx**: I2C Instance.

### Return values

- **Value**: between Min\_Data=0x004 and Max\_Data=0xFFFF, except in FAST DUTY mode where Min\_Data=0x001.

### Reference Manual to LL API cross reference:

- CCR CCR LL\_I2C\_GetClockPeriod

### LL\_I2C\_ConfigSpeed

### Function name

```
__STATIC_INLINE void LL_I2C_ConfigSpeed (I2C_TypeDef * I2Cx, uint32_t PeriphClock, uint32_t ClockSpeed, uint32_t DutyCycle)
```

### Function description

Configure the SCL speed.

### Parameters

- **I2Cx**: I2C Instance.
- **PeriphClock**: Peripheral Clock (in Hz)
- **ClockSpeed**: This parameter must be a value lower than 400kHz (in Hz).
- **DutyCycle**: This parameter can be one of the following values:
  - LL\_I2C\_DUTYCYCLE\_2
  - LL\_I2C\_DUTYCYCLE\_16\_9

### Return values

- **None**:

### Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

### Reference Manual to LL API cross reference:

- CR2\_FREQ LL\_I2C\_ConfigSpeed
- TRISE\_TRISE LL\_I2C\_ConfigSpeed
- CCR\_FS LL\_I2C\_ConfigSpeed
- CCR\_DUTY LL\_I2C\_ConfigSpeed
- CCR\_CCR LL\_I2C\_ConfigSpeed

### 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 SMBUS LL\_I2C\_SetMode
- CR1 SMBTYPE LL\_I2C\_SetMode
- CR1 ENARP 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 SMBUS LL\_I2C\_GetMode
- CR1 SMBTYPE LL\_I2C\_GetMode
- CR1 ENARP 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 driven 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 ALERT 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 driven (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 ALERT 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 ALERT 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 ENPEC 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 ENPEC 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 ENPEC `LL_I2C_IsEnabledSMBusPEC`

**LL\_I2C\_EnableIT\_TX**

**Function name**

`__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)`

**Function description**

Enable TXE interrupt.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR2 ITEVTEN `LL_I2C_EnableIT_TX`
- CR2 ITBUFEN `LL_I2C_EnableIT_TX`

**LL\_I2C\_DisableIT\_TX**

**Function name**

`__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)`

**Function description**

Disable TXE interrupt.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR2 ITEVTEN `LL_I2C_DisableIT_TX`
- CR2 ITBUFEN `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 TXE Interrupt is enabled or disabled.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **State:** of bit (1 or 0).



**Reference Manual to LL API cross reference:**

- CR2 ITEVTEN LL\_I2C\_IsEnabledIT\_TX
- CR2 ITBUFEN 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:**

- CR2 ITEVTEN LL\_I2C\_EnableIT\_RX
- CR2 ITBUFEN 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:**

- CR2 ITEVTEN LL\_I2C\_DisableIT\_RX
- CR2 ITBUFEN 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:**

- CR2 ITEVTEN LL\_I2C\_IsEnabledIT\_RX
- CR2 ITBUFEN LL\_I2C\_IsEnabledIT\_RX

### LL\_I2C\_EnableIT\_EVT

#### Function name

```
__STATIC_INLINE void LL_I2C_EnableIT_EVT (I2C_TypeDef * I2Cx)
```

#### Function description

Enable Events interrupts.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **None**:

#### Notes

- Any of these events will generate interrupt : Start Bit (SB) Address sent, Address matched (ADDR) 10-bit header sent (ADD10) Stop detection (STOPF) Byte transfer finished (BTF)
- Any of these events will generate interrupt if Buffer interrupts are enabled too(using unitary function LL\_I2C\_EnableIT\_BUF()) : Receive buffer not empty (RXNE) Transmit buffer empty (TXE)

#### Reference Manual to LL API cross reference:

- CR2 ITEVTEN LL\_I2C\_EnableIT\_EVT

### LL\_I2C\_DisableIT\_EVT

#### Function name

```
__STATIC_INLINE void LL_I2C_DisableIT_EVT (I2C_TypeDef * I2Cx)
```

#### Function description

Disable Events interrupts.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **None**:

#### Notes

- Any of these events will generate interrupt : Start Bit (SB) Address sent, Address matched (ADDR) 10-bit header sent (ADD10) Stop detection (STOPF) Byte transfer finished (BTF) Receive buffer not empty (RXNE) Transmit buffer empty (TXE)

#### Reference Manual to LL API cross reference:

- CR2 ITEVTEN LL\_I2C\_DisableIT\_EVT

### LL\_I2C\_IsEnabledIT\_EVT

#### Function name

```
__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_EVT (I2C_TypeDef * I2Cx)
```

#### Function description

Check if Events interrupts are enabled or disabled.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **State**: of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR2 ITEVTEN LL\_I2C\_IsEnabledIT\_EVT

**LL\_I2C\_EnableIT\_BUF**
**Function name**

```
__STATIC_INLINE void LL_I2C_EnableIT_BUF (I2C_TypeDef * I2Cx)
```

**Function description**

Enable Buffer interrupts.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

**Notes**

- Any of these Buffer events will generate interrupt if Events interrupts are enabled too(using unitary function LL\_I2C\_EnableIT\_EVT()) : Receive buffer not empty (RXNE) Transmit buffer empty (TXE)

**Reference Manual to LL API cross reference:**

- CR2 ITBUFEN LL\_I2C\_EnableIT\_BUF

**LL\_I2C\_DisableIT\_BUF**
**Function name**

```
__STATIC_INLINE void LL_I2C_DisableIT_BUF (I2C_TypeDef * I2Cx)
```

**Function description**

Disable Buffer interrupts.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

**Notes**

- Any of these Buffer events will generate interrupt : Receive buffer not empty (RXNE) Transmit buffer empty (TXE)

**Reference Manual to LL API cross reference:**

- CR2 ITBUFEN LL\_I2C\_DisableIT\_BUF

**LL\_I2C\_IsEnabledIT\_BUF**
**Function name**

```
__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_BUF (I2C_TypeDef * I2Cx)
```

**Function description**

Check if Buffer interrupts are enabled or disabled.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR2 ITBUFEN LL\_I2C\_IsEnabledIT\_BUF

**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 : Bus Error detection (BERR) Arbitration Loss (ARLO) Acknowledge Failure(AF) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (SMBALERT)

**Reference Manual to LL API cross reference:**

- CR2 ITERREN 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 : Bus Error detection (BERR) Arbitration Loss (ARLO) Acknowledge Failure(AF) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (SMBALERT)

**Reference Manual to LL API cross reference:**

- CR2 ITERREN 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:**

- CR2 ITERREN 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:**

- SR1 TXE LL\_I2C\_IsActiveFlag\_TXE

**LL\_I2C\_IsActiveFlag\_BTF**
**Function name**

```
__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BTF (I2C_TypeDef * I2Cx)
```

**Function description**

Indicate the status of Byte Transfer Finished flag.

**Parameters**

- **I2Cx**: I2C Instance.

**Return values**

- **State**: of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- SR1 BTF LL\_I2C\_IsActiveFlag\_BTF

**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:

- SR1 RXNE LL\_I2C\_IsActiveFlag\_RXNE

### LL\_I2C\_IsActiveFlag\_SB

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_SB (I2C_TypeDef * I2Cx)`

### Function description

Indicate the status of Start Bit (master mode).

### Parameters

- **I2Cx:** I2C Instance.

### Return values

- **State:** of bit (1 or 0).

### Notes

- RESET: When No Start condition. SET: When Start condition is generated.

### Reference Manual to LL API cross reference:

- SR1 SB LL\_I2C\_IsActiveFlag\_SB

### LL\_I2C\_IsActiveFlag\_ADDR

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)`

### Function description

Indicate the status of Address sent (master mode) or 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 address is fully sent (master mode) or when the received slave address matched with one of the enabled slave address (slave mode).

### Reference Manual to LL API cross reference:

- SR1 ADDR LL\_I2C\_IsActiveFlag\_ADDR

### LL\_I2C\_IsActiveFlag\_ADD10

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADD10 (I2C_TypeDef * I2Cx)`

### Function description

Indicate the status of 10-bit header sent (master mode).

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Notes

- RESET: When no ADD10 event occurred. SET: When the master has sent the first address byte (header).

#### Reference Manual to LL API cross reference:

- SR1 ADD10 LL\_I2C\_IsActiveFlag\_ADD10

#### LL\_I2C\_IsActiveFlag\_AF

#### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_AF (I2C_TypeDef * I2Cx)`

#### Function description

Indicate the status of Acknowledge failure flag.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Notes

- RESET: No acknowledge failure. SET: When an acknowledge failure is received after a byte transmission.

#### Reference Manual to LL API cross reference:

- SR1 AF LL\_I2C\_IsActiveFlag\_AF

#### 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 (slave mode).

#### 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:

- SR1 STOPF LL\_I2C\_IsActiveFlag\_STOP

#### 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:

- SR1 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:

- SR1 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.

### 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:

- SR1 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.

#### Reference Manual to LL API cross reference:

- SR1 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.

#### Reference Manual to LL API cross reference:

- SR1 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.

**Reference Manual to LL API cross reference:**

- SR1 SMBALERT 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:**

- SR2 BUSY LL\_I2C\_IsActiveFlag\_BUSY

**LL\_I2C\_IsActiveFlag\_DUAL**
**Function name**

```
__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_DUAL (I2C_TypeDef * I2Cx)
```

**Function description**

Indicate the status of Dual flag.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Notes**

- RESET: Received address matched with OAR1. SET: Received address matched with OAR2.

**Reference Manual to LL API cross reference:**

- SR2 DUALF LL\_I2C\_IsActiveFlag\_DUAL

**LL\_I2C\_IsActiveSMBusFlag\_SMBHOST**
**Function name**

```
__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_SMBHOST (I2C_TypeDef * I2Cx)
```

**Function description**

Indicate the status of SMBus Host address reception (Slave mode).

**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: No SMBus Host address SET: SMBus Host address received.
- This status is cleared by hardware after a STOP condition or repeated START condition.

### Reference Manual to LL API cross reference:

- SR2 SMBHOST LL\_I2C\_IsActiveSMBusFlag\_SMBHOST

#### LL\_I2C\_IsActiveSMBusFlag\_SMBDEFAULT

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_SMBDEFAULT (I2C_TypeDef * I2Cx)`

### Function description

Indicate the status of SMBus Device default address reception (Slave mode).

### 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: No SMBus Device default address SET: SMBus Device default address received.
- This status is cleared by hardware after a STOP condition or repeated START condition.

### Reference Manual to LL API cross reference:

- SR2 SMBDEFAULT LL\_I2C\_IsActiveSMBusFlag\_SMBDEFAULT

#### LL\_I2C\_IsActiveFlag\_GENCALL

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_GENCALL (I2C_TypeDef * I2Cx)`

### Function description

Indicate the status of General call address reception (Slave mode).

### Parameters

- **I2Cx:** I2C Instance.

### Return values

- **State:** of bit (1 or 0).

### Notes

- RESET: No General call address SET: General call address received.
- This status is cleared by hardware after a STOP condition or repeated START condition.

### Reference Manual to LL API cross reference:

- SR2 GENCALL LL\_I2C\_IsActiveFlag\_GENCALL

### LL\_I2C\_IsActiveFlag\_MSL

#### Function name

```
__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_MSL (I2C_TypeDef * I2Cx)
```

#### Function description

Indicate the status of Master/Slave flag.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Notes

- RESET: Slave Mode. SET: Master Mode.

#### Reference Manual to LL API cross reference:

- SR2 MSL LL\_I2C\_IsActiveFlag\_MSL

### 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**:

#### Notes

- Clearing this flag is done by a read access to the I2Cx\_SR1 register followed by a read access to the I2Cx\_SR2 register.

#### Reference Manual to LL API cross reference:

- SR1 ADDR LL\_I2C\_ClearFlag\_ADDR

### LL\_I2C\_ClearFlag\_AF

#### Function name

```
__STATIC_INLINE void LL_I2C_ClearFlag_AF (I2C_TypeDef * I2Cx)
```

#### Function description

Clear Acknowledge failure flag.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- SR1 AF LL\_I2C\_ClearFlag\_AF

### 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:**

#### Notes

- Clearing this flag is done by a read access to the I2Cx\_SR1 register followed by a write access to I2Cx\_CR1 register.

#### Reference Manual to LL API cross reference:

- SR1 STOPF LL\_I2C\_ClearFlag\_STOP
- CR1 PE LL\_I2C\_ClearFlag\_STOP

### 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:

- SR1 BERR 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:

- SR1 ARLO 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:

- SR1 OVR 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:**

#### Reference Manual to LL API cross reference:

- SR1 PECERR 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:

- SR1 TIMEOUT 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:

- SR1 SMBALERT LL\_I2C\_ClearSMBusFlag\_ALERT

### LL\_I2C\_EnableReset

#### Function name

```
__STATIC_INLINE void LL_I2C_EnableReset (I2C_TypeDef * I2Cx)
```

#### Function description

Enable Reset of I2C peripheral.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 SWRST LL\_I2C\_EnableReset

### LL\_I2C\_DisableReset

#### Function name

```
__STATIC_INLINE void LL_I2C_DisableReset (I2C_TypeDef * I2Cx)
```

#### Function description

Disable Reset of I2C peripheral.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **None**:

#### Reference Manual to LL API cross reference:

- CR1 SWRST LL\_I2C\_DisableReset

### LL\_I2C\_IsResetEnabled

#### Function name

```
__STATIC_INLINE uint32_t LL_I2C_IsResetEnabled (I2C_TypeDef * I2Cx)
```

#### Function description

Check if the I2C peripheral is under reset state or not.

#### Parameters

- **I2Cx**: I2C Instance.

#### Return values

- **State**: of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR1 SWRST LL\_I2C\_IsResetEnabled

### 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 or Master mode.

#### Reference Manual to LL API cross reference:

- CR1 ACK 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:**

- CR1 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:**

- CR1 STOP LL\_I2C\_GenerateStopCondition

**LL\_I2C\_EnableBitPOS**

**Function name**

**\_\_STATIC\_INLINE void LL\_I2C\_EnableBitPOS (I2C\_TypeDef \* I2Cx)**

**Function description**

Enable bit POS (master/host mode).

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

**Notes**

- In that case, the ACK bit controls the (N)ACK of the next byte received or the PEC bit indicates that the next byte in shift register is a PEC.

**Reference Manual to LL API cross reference:**

- CR1 POS LL\_I2C\_EnableBitPOS

**LL\_I2C\_DisableBitPOS**

**Function name**

**\_\_STATIC\_INLINE void LL\_I2C\_DisableBitPOS (I2C\_TypeDef \* I2Cx)**

**Function description**

Disable bit POS (master/host mode).

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

### Notes

- In that case, the ACK bit controls the (N)ACK of the current byte received or the PEC bit indicates that the current byte in shift register is a PEC.

### Reference Manual to LL API cross reference:

- CR1 POS LL\_I2C\_DisableBitPOS

#### LL\_I2C\_IsEnabledBitPOS

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsEnabledBitPOS (I2C_TypeDef * I2Cx)`

### Function description

Check if bit POS is enabled or disabled.

### Parameters

- **I2Cx**: I2C Instance.

### Return values

- **State**: of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR1 POS LL\_I2C\_IsEnabledBitPOS

#### LL\_I2C\_GetTransferDirection

### Function name

`__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (I2C_TypeDef * I2Cx)`

### Function description

Indicate the value of transfer direction.

### 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: Bus is in read transfer (peripheral point of view). SET: Bus is in write transfer (peripheral point of view).

### Reference Manual to LL API cross reference:

- SR2 TRA LL\_I2C\_GetTransferDirection

#### LL\_I2C\_EnableLastDMA

### Function name

`__STATIC_INLINE void LL_I2C_EnableLastDMA (I2C_TypeDef * I2Cx)`

### Function description

Enable DMA last transfer.

### Parameters

- **I2Cx**: I2C Instance.

**Return values**

- **None:**

**Notes**

- This action mean that next DMA EOT is the last transfer.

**Reference Manual to LL API cross reference:**

- CR2 LAST LL\_I2C\_EnableLastDMA

**LL\_I2C\_DisableLastDMA**
**Function name**

```
__STATIC_INLINE void LL_I2C_DisableLastDMA (I2C_TypeDef * I2Cx)
```

**Function description**

Disable DMA last transfer.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **None:**

**Notes**

- This action mean that next DMA EOT is not the last transfer.

**Reference Manual to LL API cross reference:**

- CR2 LAST LL\_I2C\_DisableLastDMA

**LL\_I2C\_IsEnabledLastDMA**
**Function name**

```
__STATIC_INLINE uint32_t LL_I2C_IsEnabledLastDMA (I2C_TypeDef * I2Cx)
```

**Function description**

Check if DMA last transfer is enabled or disabled.

**Parameters**

- **I2Cx:** I2C Instance.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR2 LAST LL\_I2C\_IsEnabledLastDMA

**LL\_I2C\_EnableSMBusPECCCompare**
**Function name**

```
__STATIC_INLINE void LL_I2C_EnableSMBusPECCCompare (I2C_TypeDef * I2Cx)
```

**Function description**

Enable transfer or 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 compared, or by a START or STOP condition, it is also cleared by software.

### Reference Manual to LL API cross reference:

- CR1 PEC `LL_I2C_EnableSMBusPECCompare`

### **LL\_I2C\_DisableSMBusPECCompare**

### Function name

`__STATIC_INLINE void LL_I2C_DisableSMBusPECCompare (I2C_TypeDef * I2Cx)`

### Function description

Disable transfer or 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.

### Reference Manual to LL API cross reference:

- CR1 PEC `LL_I2C_DisableSMBusPECCompare`

### **LL\_I2C\_IsEnabledSMBusPECCompare**

### Function name

`__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (I2C_TypeDef * I2Cx)`

### Function description

Check if the SMBus Packet Error byte transfer or 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:

- CR1 PEC `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:

- SR2 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=0x0 and Max\_Data=0xFF

### Reference Manual to LL API cross reference:

- DR DR 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=0x0 and Max\_Data=0xFF

### Return values

- **None**:

### Reference Manual to LL API cross reference:

- DR DR LL\_I2C\_TransmitData8

### LL\_I2C\_Init

### Function name

```
uint32_t 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

**uint32\_t 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**:

## 83.3 I2C Firmware driver defines

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

### 83.3.1 I2C

I2C

#### **Analog Filter Selection**

#### LL\_I2C\_ANALOGFILTER\_ENABLE

Analog filter is enabled.

#### LL\_I2C\_ANALOGFILTER\_DISABLE

Analog filter is disabled.

#### **Master Clock Speed Mode**

#### LL\_I2C\_CLOCK\_SPEED\_STANDARD\_MODE

Master clock speed range is standard mode

#### LL\_I2C\_CLOCK\_SPEED\_FAST\_MODE

Master clock speed range is fast mode

**Read Write Direction**

#### LL\_I2C\_DIRECTION\_WRITE

Bus is in write transfer

#### LL\_I2C\_DIRECTION\_READ

Bus is in read transfer

**Fast Mode Duty Cycle**

#### LL\_I2C\_DUTYCYCLE\_2

I2C fast mode Tlow/Thigh = 2

#### LL\_I2C\_DUTYCYCLE\_16\_9

I2C fast mode Tlow/Thigh = 16/9

**Get Flags Defines**

#### LL\_I2C\_SR1\_SB

Start Bit (master mode)

#### LL\_I2C\_SR1\_ADDR

Address sent (master mode) or Address matched flag (slave mode)

#### LL\_I2C\_SR1\_BTF

Byte Transfer Finished flag

#### LL\_I2C\_SR1\_ADD10

10-bit header sent (master mode)

#### LL\_I2C\_SR1\_STOPF

Stop detection flag (slave mode)

#### LL\_I2C\_SR1\_RXNE

Data register not empty (receivers)

#### LL\_I2C\_SR1\_TXE

Data register empty (transmitters)

#### LL\_I2C\_SR1\_BERR

Bus error

#### LL\_I2C\_SR1\_ARLO

Arbitration lost

#### LL\_I2C\_SR1\_AF

Acknowledge failure flag

#### LL\_I2C\_SR1\_OVR

Overrun/Underrun

#### LL\_I2C\_SR1\_PECERR

PEC Error in reception (SMBus mode)

#### LL\_I2C\_SR1\_TIMEOUT

Timeout detection flag (SMBus mode)

#### LL\_I2C\_SR1\_SMALERT

SMBus alert (SMBus mode)

#### LL\_I2C\_SR2\_MSL

Master/Slave flag

#### LL\_I2C\_SR2\_BUSY

Bus busy flag

#### LL\_I2C\_SR2\_TRA

Transmitter/receiver direction

#### LL\_I2C\_SR2\_GENCALL

General call address (Slave mode)

#### LL\_I2C\_SR2\_SMBDEFAULT

SMBus Device default address (Slave mode)

#### LL\_I2C\_SR2\_SMBHOST

SMBus Host address (Slave mode)

#### LL\_I2C\_SR2\_DUALF

Dual flag (Slave mode)

**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\_CR2\_ITEVTEN

Events interrupts enable

#### LL\_I2C\_CR2\_ITBUFEN

Buffer interrupts enable

#### LL\_I2C\_CR2\_ITERREN

Error interrupts enable

**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.

**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

**Exported\_Macros\_Helper**

## \_\_LL\_I2C\_FREQ\_HZ\_TO\_MHZ

### Description:

- Convert Peripheral Clock Frequency in Mhz.

### Parameters:

- `__PCLK__`: This parameter must be a value of peripheral clock (in Hz).

### Return value:

- Value: of peripheral clock (in Mhz)

## \_\_LL\_I2C\_FREQ\_MHZ\_TO\_HZ

### Description:

- Convert Peripheral Clock Frequency in Hz.

### Parameters:

- `__PCLK__`: This parameter must be a value of peripheral clock (in Mhz).

### Return value:

- Value: of peripheral clock (in Hz)

## \_\_LL\_I2C\_RISE\_TIME

### Description:

- Compute I2C Clock rising time.

### Parameters:

- `__FREQRANGE__`: This parameter must be a value of peripheral clock (in Mhz).
- `__SPEED__`: This parameter must be a value lower than 400kHz (in Hz).

### Return value:

- Value: between `Min_Data=0x02` and `Max_Data=0x3F`

## \_\_LL\_I2C\_SPEED\_TO\_CCR

### Description:

- Compute Speed clock range to a Clock Control Register (`I2C_CCR_CCR`) value.

### Parameters:

- `__PCLK__`: This parameter must be a value of peripheral clock (in Hz).
- `__SPEED__`: This parameter must be a value lower than 400kHz (in Hz).
- `__DUTYCYCLE__`: This parameter can be one of the following values:
  - `LL_I2C_DUTYCYCLE_2`
  - `LL_I2C_DUTYCYCLE_16_9`

### Return value:

- Value: between `Min_Data=0x004` and `Max_Data=0xFFFF`, except in FAST DUTY mode where `Min_Data=0x001`.

## \_\_LL\_I2C\_SPEED\_STANDARD\_TO\_CCR

### Description:

- Compute Speed Standard clock range to a Clock Control Register (`I2C_CCR_CCR`) value.

### Parameters:

- `__PCLK__`: This parameter must be a value of peripheral clock (in Hz).
- `__SPEED__`: This parameter must be a value lower than 100kHz (in Hz).

### Return value:

- Value: between `Min_Data=0x004` and `Max_Data=0xFFFF`.

### `__LL_I2C_SPEED_FAST_TO_CCR`

**Description:**

- Compute Speed Fast clock range to a Clock Control Register (I2C\_CCR\_CCR) value.

**Parameters:**

- `__PCLK__`: This parameter must be a value of peripheral clock (in Hz).
- `__SPEED__`: This parameter must be a value between `Min_Data=100Khz` and `Max_Data=400Khz` (in Hz).
- `__DUTYCYCLE__`: This parameter can be one of the following values:
  - `LL_I2C_DUTYCYCLE_2`
  - `LL_I2C_DUTYCYCLE_16_9`

**Return value:**

- Value: between `Min_Data=0x001` and `Max_Data=0xFFFF`

### `__LL_I2C_10BIT_ADDRESS`

**Description:**

- Get the Least significant bits of a 10-Bits address.

**Parameters:**

- `__ADDRESS__`: This parameter must be a value of a 10-Bits slave address.

**Return value:**

- Value: between `Min_Data=0x00` and `Max_Data=0xFF`

### `__LL_I2C_10BIT_HEADER_WRITE`

**Description:**

- Convert a 10-Bits address to a 10-Bits header with Write direction.

**Parameters:**

- `__ADDRESS__`: This parameter must be a value of a 10-Bits slave address.

**Return value:**

- Value: between `Min_Data=0xF0` and `Max_Data=0xF6`

### `__LL_I2C_10BIT_HEADER_READ`

**Description:**

- Convert a 10-Bits address to a 10-Bits header with Read direction.

**Parameters:**

- `__ADDRESS__`: This parameter must be a value of a 10-Bits slave address.

**Return value:**

- Value: between `Min_Data=0xF1` and `Max_Data=0xF7`

**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

## 84 LL IWDG Generic Driver

### 84.1 IWDG Firmware driver API description

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

#### 84.1.1 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\_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\_IsReady**

### Function name

`__STATIC_INLINE uint32_t LL_IWDG_IsReady (IWDG_TypeDef * IWDGx)`

### Function description

Check if flags Prescaler & Reload 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 RVU LL\_IWDG\_IsReady

## 84.2 IWDG Firmware driver defines

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

### 84.2.1 IWDG

IWDG

***Get Flags Defines***

#### LL\_IWDG\_SR\_PVU

Watchdog prescaler value update

## LL\_IWDG\_SR\_RVU

Watchdog counter reload 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



## 85 LL LPTIM Generic Driver

### 85.1 LPTIM Firmware driver registers structures

#### 85.1.1 LL\_LPTIM\_InitTypeDef

*LL\_LPTIM\_InitTypeDef* is defined in the `stm32f4xx_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()`.

### 85.2 LPTIM Firmware driver API description

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

#### 85.2.1 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\_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 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\_PAD\_AF
  - LL\_LPTIM\_INPUT1\_SRC\_PAD\_PA4
  - LL\_LPTIM\_INPUT1\_SRC\_PAD\_PB9
  - LL\_LPTIM\_INPUT1\_SRC\_TIM\_DAC

#### Return values

- **None:**

### Reference Manual to LL API cross reference:

- OR OR LL\_LPTIM\_SetInput1Src

### 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\_TIM1\_TRGO
  - LL\_LPTIM\_TRIG\_SOURCE\_TIM5\_TRGO
- **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\_TIM1\_TRGO
  - LL\_LPTIM\_TRIG\_SOURCE\_TIM5\_TRGO

### 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

Inform 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 CPMIE 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

## 85.3 LPTIM Firmware driver defines

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

### 85.3.1 LPTIM

LPTIM

***Input1 Source***

**LL\_LPTIM\_INPUT1\_SRC\_PAD\_AF**

**LL\_LPTIM\_INPUT1\_SRC\_PAD\_PA4**

**LL\_LPTIM\_INPUT1\_SRC\_PAD\_PB9**

**LL\_LPTIM\_INPUT1\_SRC\_TIM\_DAC**

***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 Timer 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 CONTINUOUS 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\_TIM1\_TRGO

External input trigger is connected to TIM1

#### LL\_LPTIM\_TRIG\_SOURCE\_TIM5\_TRGO

External input trigger is connected to TIM5

#### **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



## 86 LL PWR Generic Driver

### 86.1 PWR Firmware driver API description

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

#### 86.1.1 Detailed description of functions

##### LL\_PWR\_EnableUnderDriveMode

###### Function name

`__STATIC_INLINE void LL_PWR_EnableUnderDriveMode (void )`

###### Function description

Enable Under Drive Mode.

###### Return values

- **None:**

###### Notes

- This mode is enabled only with STOP low power mode. In this mode, the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main Regulator or the low power Regulator is in low voltage mode.
- If the Under-drive mode was enabled, it is automatically disabled after exiting Stop mode. When the voltage Regulator operates in Under-drive mode, an additional startup delay is induced when waking up from Stop mode.

###### Reference Manual to LL API cross reference:

- CR UDEN LL\_PWR\_EnableUnderDriveMode

##### LL\_PWR\_DisableUnderDriveMode

###### Function name

`__STATIC_INLINE void LL_PWR_DisableUnderDriveMode (void )`

###### Function description

Disable Under Drive Mode.

###### Return values

- **None:**

###### Reference Manual to LL API cross reference:

- CR UDEN LL\_PWR\_DisableUnderDriveMode

##### LL\_PWR\_IsEnabledUnderDriveMode

###### Function name

`__STATIC_INLINE uint32_t LL_PWR_IsEnabledUnderDriveMode (void )`

###### Function description

Check if Under Drive Mode is enabled.

###### Return values

- **State:** of bit (1 or 0).

###### Reference Manual to LL API cross reference:

- CR UDEN LL\_PWR\_IsEnabledUnderDriveMode

### LL\_PWR\_EnableOverDriveSwitching

#### Function name

`__STATIC_INLINE void LL_PWR_EnableOverDriveSwitching (void )`

#### Function description

Enable Over drive switching.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR ODSWEN LL\_PWR\_EnableOverDriveSwitching

### LL\_PWR\_DisableOverDriveSwitching

#### Function name

`__STATIC_INLINE void LL_PWR_DisableOverDriveSwitching (void )`

#### Function description

Disable Over drive switching.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR ODSWEN LL\_PWR\_DisableOverDriveSwitching

### LL\_PWR\_IsEnabledOverDriveSwitching

#### Function name

`__STATIC_INLINE uint32_t LL_PWR_IsEnabledOverDriveSwitching (void )`

#### Function description

Check if Over drive switching is enabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR ODSWEN LL\_PWR\_IsEnabledOverDriveSwitching

### LL\_PWR\_EnableOverDriveMode

#### Function name

`__STATIC_INLINE void LL_PWR_EnableOverDriveMode (void )`

#### Function description

Enable Over drive Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR ODEN LL\_PWR\_EnableOverDriveMode

### LL\_PWR\_DisableOverDriveMode

#### Function name

**\_\_STATIC\_INLINE void LL\_PWR\_DisableOverDriveMode (void )**

#### Function description

Disable Over drive Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR ODEN LL\_PWR\_DisableOverDriveMode

### LL\_PWR\_IsEnabledOverDriveMode

#### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_IsEnabledOverDriveMode (void )**

#### Function description

Check if Over drive switching is enabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR ODEN LL\_PWR\_IsEnabledOverDriveMode

### LL\_PWR\_EnableMainRegulatorDeepSleepUDMode

#### Function name

**\_\_STATIC\_INLINE void LL\_PWR\_EnableMainRegulatorDeepSleepUDMode (void )**

#### Function description

Enable Main Regulator in deepsleep under-drive Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR MRUDS LL\_PWR\_EnableMainRegulatorDeepSleepUDMode

### LL\_PWR\_DisableMainRegulatorDeepSleepUDMode

#### Function name

**\_\_STATIC\_INLINE void LL\_PWR\_DisableMainRegulatorDeepSleepUDMode (void )**

#### Function description

Disable Main Regulator in deepsleep under-drive Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR MRUDS LL\_PWR\_DisableMainRegulatorDeepSleepUDMode

### LL\_PWR\_IsEnabledMainRegulatorDeepSleepUDMode

**Function name**

`__STATIC_INLINE uint32_t LL_PWR_IsEnabledMainRegulatorDeepSleepUDMode (void )`

**Function description**

Check if Main Regulator in deepsleep under-drive Mode is enabled.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR MRUDS LL\_PWR\_IsEnabledMainRegulatorDeepSleepUDMode

### LL\_PWR\_EnableLowPowerRegulatorDeepSleepUDMode

**Function name**

`__STATIC_INLINE void LL_PWR_EnableLowPowerRegulatorDeepSleepUDMode (void )`

**Function description**

Enable Low Power Regulator in deepsleep under-drive Mode.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR LPUDS LL\_PWR\_EnableLowPowerRegulatorDeepSleepUDMode

### LL\_PWR\_DisableLowPowerRegulatorDeepSleepUDMode

**Function name**

`__STATIC_INLINE void LL_PWR_DisableLowPowerRegulatorDeepSleepUDMode (void )`

**Function description**

Disable Low Power Regulator in deepsleep under-drive Mode.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR LPUDS LL\_PWR\_DisableLowPowerRegulatorDeepSleepUDMode

### LL\_PWR\_IsEnabledLowPowerRegulatorDeepSleepUDMode

**Function name**

`__STATIC_INLINE uint32_t LL_PWR_IsEnabledLowPowerRegulatorDeepSleepUDMode (void )`

**Function description**

Check if Low Power Regulator in deepsleep under-drive Mode is enabled.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR LPUDS LL\_PWR\_IsEnabledLowPowerRegulatorDeepSleepUDMode

### LL\_PWR\_EnableMainRegulatorLowVoltageMode

#### Function name

```
__STATIC_INLINE void LL_PWR_EnableMainRegulatorLowVoltageMode (void )
```

#### Function description

Enable Main Regulator low voltage Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR MRLVDS LL\_PWR\_EnableMainRegulatorLowVoltageMode

### LL\_PWR\_DisableMainRegulatorLowVoltageMode

#### Function name

```
__STATIC_INLINE void LL_PWR_DisableMainRegulatorLowVoltageMode (void )
```

#### Function description

Disable Main Regulator low voltage Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR MRLVDS LL\_PWR\_DisableMainRegulatorLowVoltageMode

### LL\_PWR\_IsEnabledMainRegulatorLowVoltageMode

#### Function name

```
__STATIC_INLINE uint32_t LL_PWR_IsEnabledMainRegulatorLowVoltageMode (void )
```

#### Function description

Check if Main Regulator low voltage Mode is enabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR MRLVDS LL\_PWR\_IsEnabledMainRegulatorLowVoltageMode

### LL\_PWR\_EnableLowPowerRegulatorLowVoltageMode

#### Function name

```
__STATIC_INLINE void LL_PWR_EnableLowPowerRegulatorLowVoltageMode (void )
```

#### Function description

Enable Low Power Regulator low voltage Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR LPLVDS LL\_PWR\_EnableLowPowerRegulatorLowVoltageMode

### LL\_PWR\_DisableLowPowerRegulatorLowVoltageMode

#### Function name

```
__STATIC_INLINE void LL_PWR_DisableLowPowerRegulatorLowVoltageMode (void )
```

#### Function description

Disable Low Power Regulator low voltage Mode.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR LPLVDS LL\_PWR\_DisableLowPowerRegulatorLowVoltageMode

### LL\_PWR\_IsEnabledLowPowerRegulatorLowVoltageMode

#### Function name

```
__STATIC_INLINE uint32_t LL_PWR_IsEnabledLowPowerRegulatorLowVoltageMode (void )
```

#### Function description

Check if Low Power Regulator low voltage Mode is enabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CR LPLVDS LL\_PWR\_IsEnabledLowPowerRegulatorLowVoltageMode

### 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
  - LL\_PWR\_REGU\_VOLTAGE\_SCALE3 (\*) LL\_PWR\_REGU\_VOLTAGE\_SCALE1 is not available for STM32F401xx devices

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR 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
  - LL\_PWR\_REGU\_VOLTAGE\_SCALE3 (\*) LL\_PWR\_REGU\_VOLTAGE\_SCALE1 is not available for STM32F401xx devices

**Reference Manual to LL API cross reference:**

- CR VOS LL\_PWR\_GetRegulVoltageScaling

**LL\_PWR\_EnableFlashPowerDown**
**Function name**

```
__STATIC_INLINE void LL_PWR_EnableFlashPowerDown (void )
```

**Function description**

Enable the Flash Power Down in Stop Mode.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR FPDS LL\_PWR\_EnableFlashPowerDown

**LL\_PWR\_DisableFlashPowerDown**
**Function name**

```
__STATIC_INLINE void LL_PWR_DisableFlashPowerDown (void )
```

**Function description**

Disable the Flash Power Down in Stop Mode.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR FPDS LL\_PWR\_DisableFlashPowerDown

**LL\_PWR\_IsEnabledFlashPowerDown**
**Function name**

```
__STATIC_INLINE uint32_t LL_PWR_IsEnabledFlashPowerDown (void )
```

**Function description**

Check if the Flash Power Down in Stop Mode is enabled.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR FPDS LL\_PWR\_IsEnabledFlashPowerDown

**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:

- CR 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:

- CR 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:

- CR DBP LL\_PWR\_IsEnabledBkUpAccess

**LL\_PWR\_EnableBkUpRegulator**

### Function name

**\_\_STATIC\_INLINE void LL\_PWR\_EnableBkUpRegulator (void )**

### Function description

Enable the backup Regulator.

### Return values

- **None:**

### Notes

- The BRE bit of the PWR\_CSR register is protected against parasitic write access. The LL\_PWR\_EnableBkUpAccess() must be called before using this API.

### Reference Manual to LL API cross reference:

- CSR BRE LL\_PWR\_EnableBkUpRegulator



### LL\_PWR\_DisableBkUpRegulator

#### Function name

```
__STATIC_INLINE void LL_PWR_DisableBkUpRegulator (void )
```

#### Function description

Disable the backup Regulator.

#### Return values

- **None:**

#### Notes

- The BRE bit of the PWR\_CSR register is protected against parasitic write access. The LL\_PWR\_EnableBkUpAccess() must be called before using this API.

#### Reference Manual to LL API cross reference:

- CSR BRE LL\_PWR\_DisableBkUpRegulator

### LL\_PWR\_IsEnabledBkUpRegulator

#### Function name

```
__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpRegulator (void )
```

#### Function description

Check if the backup Regulator is enabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CSR BRE LL\_PWR\_IsEnabledBkUpRegulator

### LL\_PWR\_SetRegulModeDS

#### Function name

```
__STATIC_INLINE void LL_PWR_SetRegulModeDS (uint32_t RegulMode)
```

#### Function description

Set voltage Regulator mode during deep sleep mode.

#### Parameters

- **RegulMode:** This parameter can be one of the following values:
  - LL\_PWR\_REGU\_DSMODE\_MAIN
  - LL\_PWR\_REGU\_DSMODE\_LOW\_POWER

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR LPDS LL\_PWR\_SetRegulModeDS

### LL\_PWR\_GetRegulModeDS

#### Function name

```
__STATIC_INLINE uint32_t LL_PWR_GetRegulModeDS (void )
```

### Function description

Get voltage Regulator mode during deep sleep mode.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_PWR\_REGU\_DSMODE\_MAIN
  - LL\_PWR\_REGU\_DSMODE\_LOW\_POWER

### Reference Manual to LL API cross reference:

- CR LPDS LL\_PWR\_GetRegulModeDS

### LL\_PWR\_SetPowerMode

### Function name

**\_\_STATIC\_INLINE void LL\_PWR\_SetPowerMode (uint32\_t PDMode)**

### Function description

Set Power Down mode when CPU enters deepsleep.

### Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL\_PWR\_MODE\_STOP\_MAINREGU
  - LL\_PWR\_MODE\_STOP\_LPREGU
  - LL\_PWR\_MODE\_STOP\_MAINREGU\_UNDERDRIVE (\*)
  - LL\_PWR\_MODE\_STOP\_LPREGU\_UNDERDRIVE (\*)
  - LL\_PWR\_MODE\_STOP\_MAINREGU\_DEEPSLEEP (\*)
  - LL\_PWR\_MODE\_STOP\_LPREGU\_DEEPSLEEP (\*)
 (\*) not available on all devices
  - LL\_PWR\_MODE\_STANDBY

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR PDDS LL\_PWR\_SetPowerMode
- 
- CR MRUDS LL\_PWR\_SetPowerMode
- 
- CR LPUDS LL\_PWR\_SetPowerMode
- 
- CR FPDS LL\_PWR\_SetPowerMode
- 
- CR MRLVDS LL\_PWR\_SetPowerMode
- 
- CR LPIVDS LL\_PWR\_SetPowerMode
- 
- CR FPDS LL\_PWR\_SetPowerMode
- 
- CR LPDS LL\_PWR\_SetPowerMode

### LL\_PWR\_GetPowerMode

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_GetPowerMode (void )**

### Function description

Get Power Down mode when CPU enters deepsleep.

### Return values

- **Returned:** value can be one of the following values:
    - LL\_PWR\_MODE\_STOP\_MAINREGU
    - LL\_PWR\_MODE\_STOP\_LPREGU
    - LL\_PWR\_MODE\_STOP\_MAINREGU\_UNDERDRIVE (\*)
    - LL\_PWR\_MODE\_STOP\_LPREGU\_UNDERDRIVE (\*)
    - LL\_PWR\_MODE\_STOP\_MAINREGU\_DEEPSLEEP (\*)
    - LL\_PWR\_MODE\_STOP\_LPREGU\_DEEPSLEEP (\*)
- (\*) not available on all devices
- LL\_PWR\_MODE\_STANDBY

### Reference Manual to LL API cross reference:

- CR PDDS LL\_PWR\_GetPowerMode
- 
- CR MRUDS LL\_PWR\_GetPowerMode
- 
- CR LPUDS LL\_PWR\_GetPowerMode
- 
- CR FPDS LL\_PWR\_GetPowerMode
- 
- CR MRLVDS LL\_PWR\_GetPowerMode
- 
- CR LPLVDS LL\_PWR\_GetPowerMode
- 
- CR FPDS LL\_PWR\_GetPowerMode
- 
- CR LPDS LL\_PWR\_GetPowerMode

### 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:**

- CR 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:**

- CR 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:**

- CR 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:**

- CR 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:

- CR PVDE LL\_PWR\_IsEnabledPVD

### 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 (\*)
 (\*) not available on all devices

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CSR EWUP LL\_PWR\_EnableWakeUpPin
- 
- CSR EWUP1 LL\_PWR\_EnableWakeUpPin
- 
- CSR EWUP2 LL\_PWR\_EnableWakeUpPin
- 
- CSR EWUP3 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 (\*)
 (\*) not available on all devices

### Return values

- **None:**

**Reference Manual to LL API cross reference:**

- CSR EWUP LL\_PWR\_DisableWakeUpPin
- 
- CSR EWUP1 LL\_PWR\_DisableWakeUpPin
- 
- CSR EWUP2 LL\_PWR\_DisableWakeUpPin
- 
- CSR EWUP3 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 (\*)
 (\*) not available on all devices

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CSR EWUP LL\_PWR\_IsEnabledWakeUpPin
- 
- CSR EWUP1 LL\_PWR\_IsEnabledWakeUpPin
- 
- CSR EWUP2 LL\_PWR\_IsEnabledWakeUpPin
- 
- CSR EWUP3 LL\_PWR\_IsEnabledWakeUpPin

**LL\_PWR\_IsActiveFlag\_WU**

**Function name**

`__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU (void )`

**Function description**

Get Wake-up Flag.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CSR WUF LL\_PWR\_IsActiveFlag\_WU

**LL\_PWR\_IsActiveFlag\_SB**

**Function name**

`__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void )`

**Function description**

Get Standby Flag.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CSR SBF LL\_PWR\_IsActiveFlag\_SB

**LL\_PWR\_IsActiveFlag\_BRR**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_IsActiveFlag\_BRR (void )**

**Function description**

Get Backup Regulator ready Flag.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CSR BRR LL\_PWR\_IsActiveFlag\_BRR

**LL\_PWR\_IsActiveFlag\_PVDO**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_IsActiveFlag\_PVDO (void )**

**Function description**

Indicate whether VDD voltage is below the selected PVD threshold.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CSR 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:**

- CSR VOS LL\_PWR\_IsActiveFlag\_VOS

**LL\_PWR\_IsActiveFlag\_OD**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_IsActiveFlag\_OD (void )**

### Function description

Indicate whether the Over-Drive mode is ready or not.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CSR ODRDY LL\_PWR\_IsActiveFlag\_OD

**LL\_PWR\_IsActiveFlag\_ODSW**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_IsActiveFlag\_ODSW (void )**

### Function description

Indicate whether the Over-Drive mode switching is ready or not.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CSR ODSWRDY LL\_PWR\_IsActiveFlag\_ODSW

**LL\_PWR\_IsActiveFlag\_UD**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_PWR\_IsActiveFlag\_UD (void )**

### Function description

Indicate whether the Under-Drive mode is ready or not.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CSR UDRDY LL\_PWR\_IsActiveFlag\_UD

**LL\_PWR\_ClearFlag\_SB**

### Function name

**\_\_STATIC\_INLINE void LL\_PWR\_ClearFlag\_SB (void )**

### Function description

Clear Standby Flag.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR 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:**

- CR CWUF LL\_PWR\_ClearFlag\_WU

**LL\_PWR\_ClearFlag\_UD**

**Function name**

**\_\_STATIC\_INLINE void LL\_PWR\_ClearFlag\_UD (void )**

**Function description**

Clear Under-Drive ready Flag.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CSR UDRDY LL\_PWR\_ClearFlag\_UD

**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

## 86.2 PWR Firmware driver defines

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

### 86.2.1 PWR

PWR

***Clear Flags Defines***

**LL\_PWR\_CR\_CSBF**

Clear standby flag

**LL\_PWR\_CR\_CWUF**

Clear wakeup flag

***Get Flags Defines***

**LL\_PWR\_CSR\_WUF**

Wakeup flag

**LL\_PWR\_CSR\_SBF**

Standby flag

**LL\_PWR\_CSR\_PVDO**

Power voltage detector output flag

#### LL\_PWR\_CSR\_VOS

Voltage scaling select flag

#### LL\_PWR\_CSR\_EWUP1

Enable WKUP pin

**Mode Power**

#### LL\_PWR\_MODE\_STOP\_MAINREGU

Enter Stop mode when the CPU enters deepsleep

#### LL\_PWR\_MODE\_STOP\_LPREGU

Enter Stop mode (with low power Regulator ON) when the CPU enters deepsleep

#### LL\_PWR\_MODE\_STOP\_MAINREGU\_UNDERDRIVE

Enter Stop mode (with main Regulator in under-drive mode) when the CPU enters deepsleep

#### LL\_PWR\_MODE\_STOP\_LPREGU\_UNDERDRIVE

Enter Stop mode (with low power Regulator in under-drive mode) when the CPU enters deepsleep

#### LL\_PWR\_MODE\_STOP\_MAINREGU\_DEEPSLEEP

Enter Stop mode (with main Regulator in Deep Sleep mode) when the CPU enters deepsleep

#### LL\_PWR\_MODE\_STOP\_LPREGU\_DEEPSLEEP

Enter Stop mode (with low power Regulator in Deep Sleep mode) when the CPU enters deepsleep

#### LL\_PWR\_MODE\_STANDBY

Enter Standby mode when the CPU enters deepsleep

**Power Voltage Detector Level**

#### LL\_PWR\_PVDLEVEL\_0

Voltage threshold detected by PVD 2.2 V

#### LL\_PWR\_PVDLEVEL\_1

Voltage threshold detected by PVD 2.3 V

#### LL\_PWR\_PVDLEVEL\_2

Voltage threshold detected by PVD 2.4 V

#### LL\_PWR\_PVDLEVEL\_3

Voltage threshold detected by PVD 2.5 V

#### LL\_PWR\_PVDLEVEL\_4

Voltage threshold detected by PVD 2.6 V

#### LL\_PWR\_PVDLEVEL\_5

Voltage threshold detected by PVD 2.7 V

#### LL\_PWR\_PVDLEVEL\_6

Voltage threshold detected by PVD 2.8 V

#### LL\_PWR\_PVDLEVEL\_7

Voltage threshold detected by PVD 2.9 V

**Regulator Mode In Deep Sleep Mode**

#### LL\_PWR\_REGU\_DSMODE\_MAIN

Voltage Regulator in main mode during deepsleep mode

**LL\_PWR\_REGU\_DSMODE\_LOW\_POWER**

Voltage Regulator in low-power mode during deepsleep mode

***Regulator Voltage***

**LL\_PWR\_REGU\_VOLTAGE\_SCALE3**
**LL\_PWR\_REGU\_VOLTAGE\_SCALE2**
**LL\_PWR\_REGU\_VOLTAGE\_SCALE1**

***Wakeup Pins***

**LL\_PWR\_WAKEUP\_PIN1**

WKUP pin : PA0

***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

## 87 LL RCC Generic Driver

### 87.1 RCC Firmware driver registers structures

#### 87.1.1 LL\_RCC\_ClocksTypeDef

*LL\_RCC\_ClocksTypeDef* is defined in the `stm32f4xx_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

### 87.2 RCC Firmware driver API description

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

#### 87.2.1 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\_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:

- CR 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 = 31

#### 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:

- CR 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 = 31

#### Reference Manual to LL API cross reference:

- CR HSITRIM LL\_RCC\_HSI\_GetCalibTrimming

### 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\_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 LSEBYP LL\_RCC\_LSE\_IsReady

**LL\_RCC\_LSE\_EnableHighDriveMode**

**Function name**

`__STATIC_INLINE void LL_RCC_LSE_EnableHighDriveMode (void )`

**Function description**

Enable LSE high drive mode.

**Return values**

- **None:**



#### Notes

- LSE high drive mode can be enabled only when the LSE clock is disabled

#### Reference Manual to LL API cross reference:

- BDCR LSEMOD LL\_RCC\_LSE\_EnableHighDriveMode

#### LL\_RCC\_LSE\_DisableHighDriveMode

#### Function name

`__STATIC_INLINE void LL_RCC_LSE_DisableHighDriveMode (void )`

#### Function description

Disable LSE high drive mode.

#### Return values

- **None:**

#### Notes

- LSE high drive mode can be disabled only when the LSE clock is disabled

#### Reference Manual to LL API cross reference:

- BDCR LSEMOD LL\_RCC\_LSE\_DisableHighDriveMode

#### 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\_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
  - LL\_RCC\_SYS\_CLKSOURCE\_PLLR (\*)
 (\*) value not defined in all devices.

#### 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
  - LL\_RCC\_SYS\_CLKSOURCE\_STATUS\_PLLR (\*)
 (\*) value not defined in all devices.

#### 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\_HSI
  - LL\_RCC\_MCO1SOURCE\_LSE
  - LL\_RCC\_MCO1SOURCE\_HSE
  - LL\_RCC\_MCO1SOURCE\_PLLCLK
  - LL\_RCC\_MCO2SOURCE\_SYSCLK
  - LL\_RCC\_MCO2SOURCE\_PLLI2S
  - LL\_RCC\_MCO2SOURCE\_HSE
  - LL\_RCC\_MCO2SOURCE\_PLLCLK
- **MCOxPrescaler:** This parameter can be one of the following values:
  - LL\_RCC\_MCO1\_DIV\_1
  - LL\_RCC\_MCO1\_DIV\_2
  - LL\_RCC\_MCO1\_DIV\_3
  - LL\_RCC\_MCO1\_DIV\_4
  - LL\_RCC\_MCO1\_DIV\_5
  - LL\_RCC\_MCO2\_DIV\_1
  - LL\_RCC\_MCO2\_DIV\_2
  - LL\_RCC\_MCO2\_DIV\_3
  - LL\_RCC\_MCO2\_DIV\_4
  - LL\_RCC\_MCO2\_DIV\_5

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CFGR MCO1 LL\_RCC\_ConfigMCO
- CFGR MCO1PRE LL\_RCC\_ConfigMCO
- CFGR MCO2 LL\_RCC\_ConfigMCO
- CFGR MCO2PRE LL\_RCC\_ConfigMCO

### 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\_PLLSAI (\*)
  - LL\_RCC\_SAI1\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI1\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI1\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLLSRC (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLSRC (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLSRC (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- DCKCFGR SAI1SRC LL\_RCC\_SetSAIClockSource
- DCKCFGR SAI2SRC LL\_RCC\_SetSAIClockSource
- DCKCFGR SAI1ASRC LL\_RCC\_SetSAIClockSource
- DCKCFGR SAI1BSRC LL\_RCC\_SetSAIClockSource

## LL\_RCC\_SetSDIOClockSource

## Function name

`__STATIC_INLINE void LL_RCC_SetSDIOClockSource (uint32_t SDIOxSource)`

## Function description

Configure SDIO clock source.

## Parameters

- **SDIOxSource:** This parameter can be one of the following values:
  - LL\_RCC\_SDIO\_CLKSOURCE\_PLL48CLK
  - LL\_RCC\_SDIO\_CLKSOURCE\_SYSCCLK

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- DCKCFGR SDIOSEL LL\_RCC\_SetSDIOClockSource
- DCKCFGR2 SDIOSEL LL\_RCC\_SetSDIOClockSource

### LL\_RCC\_SetCK48MClockSource

#### Function name

```
__STATIC_INLINE void LL_RCC_SetCK48MClockSource (uint32_t CK48MxSource)
```

#### Function description

Configure 48Mhz domain clock source.

#### Parameters

- **CK48MxSource:** This parameter can be one of the following values:
  - LL\_RCC\_CK48M\_CLKSOURCE\_PLL
  - LL\_RCC\_CK48M\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_CK48M\_CLKSOURCE\_PLLI2S (\*)
 (\*) value not defined in all devices.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- DCKCFGR CK48MSEL LL\_RCC\_SetCK48MClockSource
- DCKCFGR2 CK48MSEL LL\_RCC\_SetCK48MClockSource

### 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\_PLL
  - LL\_RCC\_RNG\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_RNG\_CLKSOURCE\_PLLI2S (\*)
 (\*) value not defined in all devices.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- DCKCFGR CK48MSEL LL\_RCC\_SetRNGClockSource
- DCKCFGR2 CK48MSEL 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\_PLL
  - LL\_RCC\_USB\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_USB\_CLKSOURCE\_PLLI2S (\*)
 (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- DCKCFGR CK48MSEL LL\_RCC\_SetUSBClockSource
- DCKCFGR2 CK48MSEL LL\_RCC\_SetUSBClockSource

### LL\_RCC\_SetI2SClockSource

#### Function name

**\_\_STATIC\_INLINE void LL\_RCC\_SetI2SClockSource (uint32\_t Source)**

#### Function description

Configure I2S clock source.

### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_I2S1\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_I2S1\_CLKSOURCE\_PIN
  - LL\_RCC\_I2S1\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_I2S1\_CLKSOURCE\_PLLSRC (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PLLSRC (\*)
 (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CFGR I2SSRC LL\_RCC\_SetI2SClockSource
- DCKCFGR I2SSRC LL\_RCC\_SetI2SClockSource
- DCKCFGR I2S1SRC LL\_RCC\_SetI2SClockSource
- DCKCFGR I2S2SRC LL\_RCC\_SetI2SClockSource

### LL\_RCC\_SetDSIClockSource

#### Function name

**\_\_STATIC\_INLINE void LL\_RCC\_SetDSIClockSource (uint32\_t Source)**

#### Function description

Configure DSI clock source.



### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_DSI\_CLKSOURCE\_PHY
  - LL\_RCC\_DSI\_CLKSOURCE\_PLL

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- DCKCFGR DSISEL LL\_RCC\_SetDSIClockSource

### 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 (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE (\*)
 (\*) value not defined in all devices.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_SAI1\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI1\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI1\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI1\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE\_PLLSRC (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLSRC (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLSRC (\*)
 (\*) value not defined in all devices.

**Reference Manual to LL API cross reference:**

- DCKCFGR SAI1SEL LL\_RCC\_GetSAIClockSource
- DCKCFGR SAI2SEL LL\_RCC\_GetSAIClockSource
- DCKCFGR SAI1ASRC LL\_RCC\_GetSAIClockSource
- DCKCFGR SAI1BSRC LL\_RCC\_GetSAIClockSource

**LL\_RCC\_GetSDIOClockSource**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_GetSDIOClockSource (uint32_t SDIOx)`

**Function description**

Get SDIOx clock source.

**Parameters**

- **SDIOx:** This parameter can be one of the following values:
  - LL\_RCC\_SDIO\_CLKSOURCE

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_RCC\_SDIO\_CLKSOURCE\_PLL48CLK
  - LL\_RCC\_SDIO\_CLKSOURCE\_SYSCLK

**Reference Manual to LL API cross reference:**

- DCKCFGR SDIOSEL LL\_RCC\_GetSDIOClockSource
- DCKCFGR2 SDIOSEL LL\_RCC\_GetSDIOClockSource

**LL\_RCC\_GetCK48MClockSource**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_GetCK48MClockSource (uint32_t CK48Mx)`

**Function description**

Get 48Mhz domain clock source.

**Parameters**

- **CK48Mx:** This parameter can be one of the following values:
  - LL\_RCC\_CK48M\_CLKSOURCE

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_RCC\_CK48M\_CLKSOURCE\_PLL
  - LL\_RCC\_CK48M\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_CK48M\_CLKSOURCE\_PLLI2S (\*)
 (\*) value not defined in all devices.

**Reference Manual to LL API cross reference:**

- DCKCFGR CK48MSEL LL\_RCC\_GetCK48MClockSource
- DCKCFGR2 CK48MSEL LL\_RCC\_GetCK48MClockSource

**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\_PLL
  - LL\_RCC\_RNG\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_RNG\_CLKSOURCE\_PLLI2S (\*)
 (\*) value not defined in all devices.

### Reference Manual to LL API cross reference:

- DCKCFGR CK48MSEL LL\_RCC\_GetRNGClockSource
- DCKCFGR2 CK48MSEL 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\_PLL
  - LL\_RCC\_USB\_CLKSOURCE\_PLLSAI (\*)
  - LL\_RCC\_USB\_CLKSOURCE\_PLLI2S (\*)
 (\*) value not defined in all devices.

### Reference Manual to LL API cross reference:

- DCKCFGR CK48MSEL LL\_RCC\_GetUSBClockSource
- DCKCFGR2 CK48MSEL LL\_RCC\_GetUSBClockSource

### LL\_RCC\_GetI2SClockSource

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_RCC\_GetI2SClockSource (uint32\_t I2Sx)**

### Function description

Get I2S Clock Source.

### Parameters

- **I2Sx:** This parameter can be one of the following values:
  - LL\_RCC\_I2S1\_CLKSOURCE
  - LL\_RCC\_I2S2\_CLKSOURCE (\*)

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_I2S1\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_I2S1\_CLKSOURCE\_PIN
  - LL\_RCC\_I2S1\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_I2S1\_CLKSOURCE\_PLLSRC (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PLLI2S (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PIN (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PLL (\*)
  - LL\_RCC\_I2S2\_CLKSOURCE\_PLLSRC (\*)

(\*) value not defined in all devices.

### Reference Manual to LL API cross reference:

- CFGFR I2SSRC LL\_RCC\_GetI2SClockSource
- DCKCFGR I2SSRC LL\_RCC\_GetI2SClockSource
- DCKCFGR I2S1SRC LL\_RCC\_GetI2SClockSource
- DCKCFGR I2S2SRC LL\_RCC\_GetI2SClockSource

### LL\_RCC\_GetDSIClockSource

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_GetDSIClockSource (uint32_t DSIx)`

#### Function description

Get DSI Clock Source.

#### Parameters

- **DSIx:** This parameter can be one of the following values:
  - LL\_RCC\_DSI\_CLKSOURCE

#### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_DSI\_CLKSOURCE\_PHY
  - LL\_RCC\_DSI\_CLKSOURCE\_PLL

### Reference Manual to LL API cross reference:

- DCKCFGR DSISEL LL\_RCC\_GetDSIClockSource

### 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

#### 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

### 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\_SetRTC\_HSEPrescaler

### Function name

`__STATIC_INLINE void LL_RCC_SetRTC_HSEPrescaler (uint32_t Prescaler)`

### Function description

Set HSE Prescalers for RTC Clock.

## Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL\_RCC\_RTC\_NOCLOCK
  - LL\_RCC\_RTC\_HSE\_DIV\_2
  - LL\_RCC\_RTC\_HSE\_DIV\_3
  - LL\_RCC\_RTC\_HSE\_DIV\_4
  - LL\_RCC\_RTC\_HSE\_DIV\_5
  - LL\_RCC\_RTC\_HSE\_DIV\_6
  - LL\_RCC\_RTC\_HSE\_DIV\_7
  - LL\_RCC\_RTC\_HSE\_DIV\_8
  - LL\_RCC\_RTC\_HSE\_DIV\_9
  - LL\_RCC\_RTC\_HSE\_DIV\_10
  - LL\_RCC\_RTC\_HSE\_DIV\_11
  - LL\_RCC\_RTC\_HSE\_DIV\_12
  - LL\_RCC\_RTC\_HSE\_DIV\_13
  - LL\_RCC\_RTC\_HSE\_DIV\_14
  - LL\_RCC\_RTC\_HSE\_DIV\_15
  - LL\_RCC\_RTC\_HSE\_DIV\_16
  - LL\_RCC\_RTC\_HSE\_DIV\_17
  - LL\_RCC\_RTC\_HSE\_DIV\_18
  - LL\_RCC\_RTC\_HSE\_DIV\_19
  - LL\_RCC\_RTC\_HSE\_DIV\_20
  - LL\_RCC\_RTC\_HSE\_DIV\_21
  - LL\_RCC\_RTC\_HSE\_DIV\_22
  - LL\_RCC\_RTC\_HSE\_DIV\_23
  - LL\_RCC\_RTC\_HSE\_DIV\_24
  - LL\_RCC\_RTC\_HSE\_DIV\_25
  - LL\_RCC\_RTC\_HSE\_DIV\_26
  - LL\_RCC\_RTC\_HSE\_DIV\_27
  - LL\_RCC\_RTC\_HSE\_DIV\_28
  - LL\_RCC\_RTC\_HSE\_DIV\_29
  - LL\_RCC\_RTC\_HSE\_DIV\_30
  - LL\_RCC\_RTC\_HSE\_DIV\_31

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- CFGR RTCPRE LL\_RCC\_SetRTC\_HSEPrescaler

**LL\_RCC\_GetRTC\_HSEPrescaler**

## Function name

**\_\_STATIC\_INLINE uint32\_t LL\_RCC\_GetRTC\_HSEPrescaler (void )**

## Function description

Get HSE Prescalers for RTC Clock.

## Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_RTC\_NOCLOCK
  - LL\_RCC\_RTC\_HSE\_DIV\_2
  - LL\_RCC\_RTC\_HSE\_DIV\_3
  - LL\_RCC\_RTC\_HSE\_DIV\_4
  - LL\_RCC\_RTC\_HSE\_DIV\_5
  - LL\_RCC\_RTC\_HSE\_DIV\_6
  - LL\_RCC\_RTC\_HSE\_DIV\_7
  - LL\_RCC\_RTC\_HSE\_DIV\_8
  - LL\_RCC\_RTC\_HSE\_DIV\_9
  - LL\_RCC\_RTC\_HSE\_DIV\_10
  - LL\_RCC\_RTC\_HSE\_DIV\_11
  - LL\_RCC\_RTC\_HSE\_DIV\_12
  - LL\_RCC\_RTC\_HSE\_DIV\_13
  - LL\_RCC\_RTC\_HSE\_DIV\_14
  - LL\_RCC\_RTC\_HSE\_DIV\_15
  - LL\_RCC\_RTC\_HSE\_DIV\_16
  - LL\_RCC\_RTC\_HSE\_DIV\_17
  - LL\_RCC\_RTC\_HSE\_DIV\_18
  - LL\_RCC\_RTC\_HSE\_DIV\_19
  - LL\_RCC\_RTC\_HSE\_DIV\_20
  - LL\_RCC\_RTC\_HSE\_DIV\_21
  - LL\_RCC\_RTC\_HSE\_DIV\_22
  - LL\_RCC\_RTC\_HSE\_DIV\_23
  - LL\_RCC\_RTC\_HSE\_DIV\_24
  - LL\_RCC\_RTC\_HSE\_DIV\_25
  - LL\_RCC\_RTC\_HSE\_DIV\_26
  - LL\_RCC\_RTC\_HSE\_DIV\_27
  - LL\_RCC\_RTC\_HSE\_DIV\_28
  - LL\_RCC\_RTC\_HSE\_DIV\_29
  - LL\_RCC\_RTC\_HSE\_DIV\_30
  - LL\_RCC\_RTC\_HSE\_DIV\_31

## Reference Manual to LL API cross reference:

- CFGR RTCPRE LL\_RCC\_GetRTC\_HSEPrescaler

## LL\_RCC\_SetTIMPrescaler

### Function name

`__STATIC_INLINE void LL_RCC_SetTIMPrescaler (uint32_t Prescaler)`

### Function description

Set Timers Clock Prescalers.

### Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL\_RCC\_TIM\_PRESCALER\_TWICE
  - LL\_RCC\_TIM\_PRESCALER\_FOUR\_TIMES

### Return values

- **None:**



**Reference Manual to LL API cross reference:**

- DCKCFGR TIMPRE LL\_RCC\_SetTIMPrescaler

**LL\_RCC\_GetTIMPrescaler**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_GetTIMPrescaler (void )`

**Function description**

Get Timers Clock Prescalers.

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_RCC\_TIM\_PRESCALER\_TWICE
  - LL\_RCC\_TIM\_PRESCALER\_FOUR\_TIMES

**Reference Manual to LL API cross reference:**

- DCKCFGR TIMPRE LL\_RCC\_GetTIMPrescaler

**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 PLLP_R)
```

**Function description**

Configure PLL used for SYSCLK Domain.

### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLLM:** This parameter can be one of the following values:
  - 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
  - LL\_RCC\_PLLM\_DIV\_17
  - LL\_RCC\_PLLM\_DIV\_18
  - LL\_RCC\_PLLM\_DIV\_19
  - LL\_RCC\_PLLM\_DIV\_20
  - LL\_RCC\_PLLM\_DIV\_21
  - LL\_RCC\_PLLM\_DIV\_22
  - LL\_RCC\_PLLM\_DIV\_23
  - LL\_RCC\_PLLM\_DIV\_24
  - LL\_RCC\_PLLM\_DIV\_25
  - LL\_RCC\_PLLM\_DIV\_26
  - LL\_RCC\_PLLM\_DIV\_27
  - LL\_RCC\_PLLM\_DIV\_28
  - LL\_RCC\_PLLM\_DIV\_29
  - LL\_RCC\_PLLM\_DIV\_30
  - LL\_RCC\_PLLM\_DIV\_31
  - LL\_RCC\_PLLM\_DIV\_32
  - LL\_RCC\_PLLM\_DIV\_33
  - LL\_RCC\_PLLM\_DIV\_34
  - LL\_RCC\_PLLM\_DIV\_35
  - LL\_RCC\_PLLM\_DIV\_36
  - LL\_RCC\_PLLM\_DIV\_37
  - LL\_RCC\_PLLM\_DIV\_38
  - LL\_RCC\_PLLM\_DIV\_39
  - LL\_RCC\_PLLM\_DIV\_40
  - LL\_RCC\_PLLM\_DIV\_41
  - LL\_RCC\_PLLM\_DIV\_42
  - LL\_RCC\_PLLM\_DIV\_43
  - LL\_RCC\_PLLM\_DIV\_44
  - LL\_RCC\_PLLM\_DIV\_45
  - LL\_RCC\_PLLM\_DIV\_46
  - LL\_RCC\_PLLM\_DIV\_47
  - LL\_RCC\_PLLM\_DIV\_48
  - LL\_RCC\_PLLM\_DIV\_49
  - LL\_RCC\_PLLM\_DIV\_50
  - LL\_RCC\_PLLM\_DIV\_51
  - LL\_RCC\_PLLM\_DIV\_52
  - LL\_RCC\_PLLM\_DIV\_53

- **PLLN:** Between 50/192(\*) and 432
- **PLLPR:** This parameter can be one of the following values:
  - LL\_RCC\_PLLP\_DIV\_2
  - LL\_RCC\_PLLP\_DIV\_4
  - LL\_RCC\_PLLP\_DIV\_6
  - LL\_RCC\_PLLP\_DIV\_8
  - LL\_RCC\_PLLR\_DIV\_2 (\*)
  - LL\_RCC\_PLLR\_DIV\_3 (\*)
  - LL\_RCC\_PLLR\_DIV\_4 (\*)
  - LL\_RCC\_PLLR\_DIV\_5 (\*)
  - LL\_RCC\_PLLR\_DIV\_6 (\*)
  - LL\_RCC\_PLLR\_DIV\_7 (\*)

(\*) value not defined in all devices.

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLLPR 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
- PLLCFGR PLLP LL\_RCC\_PLL\_ConfigDomain\_SYS

#### **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\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLLM:** This parameter can be one of the following values:
  - 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
  - LL\_RCC\_PLLM\_DIV\_17
  - LL\_RCC\_PLLM\_DIV\_18
  - LL\_RCC\_PLLM\_DIV\_19
  - LL\_RCC\_PLLM\_DIV\_20
  - LL\_RCC\_PLLM\_DIV\_21
  - LL\_RCC\_PLLM\_DIV\_22
  - LL\_RCC\_PLLM\_DIV\_23
  - LL\_RCC\_PLLM\_DIV\_24
  - LL\_RCC\_PLLM\_DIV\_25
  - LL\_RCC\_PLLM\_DIV\_26
  - LL\_RCC\_PLLM\_DIV\_27
  - LL\_RCC\_PLLM\_DIV\_28
  - LL\_RCC\_PLLM\_DIV\_29
  - LL\_RCC\_PLLM\_DIV\_30
  - LL\_RCC\_PLLM\_DIV\_31
  - LL\_RCC\_PLLM\_DIV\_32
  - LL\_RCC\_PLLM\_DIV\_33
  - LL\_RCC\_PLLM\_DIV\_34
  - LL\_RCC\_PLLM\_DIV\_35
  - LL\_RCC\_PLLM\_DIV\_36
  - LL\_RCC\_PLLM\_DIV\_37
  - LL\_RCC\_PLLM\_DIV\_38
  - LL\_RCC\_PLLM\_DIV\_39
  - LL\_RCC\_PLLM\_DIV\_40
  - LL\_RCC\_PLLM\_DIV\_41
  - LL\_RCC\_PLLM\_DIV\_42
  - LL\_RCC\_PLLM\_DIV\_43
  - LL\_RCC\_PLLM\_DIV\_44
  - LL\_RCC\_PLLM\_DIV\_45
  - LL\_RCC\_PLLM\_DIV\_46
  - LL\_RCC\_PLLM\_DIV\_47
  - LL\_RCC\_PLLM\_DIV\_48
  - LL\_RCC\_PLLM\_DIV\_49
  - LL\_RCC\_PLLM\_DIV\_50
  - LL\_RCC\_PLLM\_DIV\_51
  - LL\_RCC\_PLLM\_DIV\_52
  - LL\_RCC\_PLLM\_DIV\_53

- **PLLN:** Between 50/192(\*) and 432
- **PLLQ:** This parameter can be one of the following values:
  - LL\_RCC\_PLLQ\_DIV\_2
  - LL\_RCC\_PLLQ\_DIV\_3
  - LL\_RCC\_PLLQ\_DIV\_4
  - LL\_RCC\_PLLQ\_DIV\_5
  - LL\_RCC\_PLLQ\_DIV\_6
  - LL\_RCC\_PLLQ\_DIV\_7
  - LL\_RCC\_PLLQ\_DIV\_8
  - LL\_RCC\_PLLQ\_DIV\_9
  - LL\_RCC\_PLLQ\_DIV\_10
  - LL\_RCC\_PLLQ\_DIV\_11
  - LL\_RCC\_PLLQ\_DIV\_12
  - LL\_RCC\_PLLQ\_DIV\_13
  - LL\_RCC\_PLLQ\_DIV\_14
  - LL\_RCC\_PLLQ\_DIV\_15

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLLQ can be written only when PLL is disabled
- This can be selected for USB, RNG, SDIO

#### 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\_ConfigDomain\_DSI

#### Function name

```
__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_DSI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)
```

#### Function description

Configure PLL used for DSI clock.



### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLLM:** This parameter can be one of the following values:
  - 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
  - LL\_RCC\_PLLM\_DIV\_17
  - LL\_RCC\_PLLM\_DIV\_18
  - LL\_RCC\_PLLM\_DIV\_19
  - LL\_RCC\_PLLM\_DIV\_20
  - LL\_RCC\_PLLM\_DIV\_21
  - LL\_RCC\_PLLM\_DIV\_22
  - LL\_RCC\_PLLM\_DIV\_23
  - LL\_RCC\_PLLM\_DIV\_24
  - LL\_RCC\_PLLM\_DIV\_25
  - LL\_RCC\_PLLM\_DIV\_26
  - LL\_RCC\_PLLM\_DIV\_27
  - LL\_RCC\_PLLM\_DIV\_28
  - LL\_RCC\_PLLM\_DIV\_29
  - LL\_RCC\_PLLM\_DIV\_30
  - LL\_RCC\_PLLM\_DIV\_31
  - LL\_RCC\_PLLM\_DIV\_32
  - LL\_RCC\_PLLM\_DIV\_33
  - LL\_RCC\_PLLM\_DIV\_34
  - LL\_RCC\_PLLM\_DIV\_35
  - LL\_RCC\_PLLM\_DIV\_36
  - LL\_RCC\_PLLM\_DIV\_37
  - LL\_RCC\_PLLM\_DIV\_38
  - LL\_RCC\_PLLM\_DIV\_39
  - LL\_RCC\_PLLM\_DIV\_40
  - LL\_RCC\_PLLM\_DIV\_41
  - LL\_RCC\_PLLM\_DIV\_42
  - LL\_RCC\_PLLM\_DIV\_43
  - LL\_RCC\_PLLM\_DIV\_44
  - LL\_RCC\_PLLM\_DIV\_45
  - LL\_RCC\_PLLM\_DIV\_46
  - LL\_RCC\_PLLM\_DIV\_47
  - LL\_RCC\_PLLM\_DIV\_48
  - LL\_RCC\_PLLM\_DIV\_49
  - LL\_RCC\_PLLM\_DIV\_50
  - LL\_RCC\_PLLM\_DIV\_51
  - LL\_RCC\_PLLM\_DIV\_52
  - LL\_RCC\_PLLM\_DIV\_53

- **PLLN:** Between 50 and 432
- **PLLr:** This parameter can be one of the following values:
  - LL\_RCC\_PLLR\_DIV\_2
  - LL\_RCC\_PLLR\_DIV\_3
  - LL\_RCC\_PLLR\_DIV\_4
  - LL\_RCC\_PLLR\_DIV\_5
  - LL\_RCC\_PLLR\_DIV\_6
  - LL\_RCC\_PLLR\_DIV\_7

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI are disabled
- PLLN/PLLr can be written only when PLL is disabled
- This can be selected for DSI

#### Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL\_RCC\_PLL\_ConfigDomain\_DSI
- PLLCFGR PLLM LL\_RCC\_PLL\_ConfigDomain\_DSI
- PLLCFGR PLLN LL\_RCC\_PLL\_ConfigDomain\_DSI
- PLLCFGR PLLR LL\_RCC\_PLL\_ConfigDomain\_DSI

#### LL\_RCC\_PLL\_ConfigDomain\_SAI

#### Function name

```
__STATIC_INLINE void LL_RCC_PLL_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)
```

#### Function description

Configure PLL used for SAI clock.

### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLLM:** This parameter can be one of the following values:
  - 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
  - LL\_RCC\_PLLM\_DIV\_17
  - LL\_RCC\_PLLM\_DIV\_18
  - LL\_RCC\_PLLM\_DIV\_19
  - LL\_RCC\_PLLM\_DIV\_20
  - LL\_RCC\_PLLM\_DIV\_21
  - LL\_RCC\_PLLM\_DIV\_22
  - LL\_RCC\_PLLM\_DIV\_23
  - LL\_RCC\_PLLM\_DIV\_24
  - LL\_RCC\_PLLM\_DIV\_25
  - LL\_RCC\_PLLM\_DIV\_26
  - LL\_RCC\_PLLM\_DIV\_27
  - LL\_RCC\_PLLM\_DIV\_28
  - LL\_RCC\_PLLM\_DIV\_29
  - LL\_RCC\_PLLM\_DIV\_30
  - LL\_RCC\_PLLM\_DIV\_31
  - LL\_RCC\_PLLM\_DIV\_32
  - LL\_RCC\_PLLM\_DIV\_33
  - LL\_RCC\_PLLM\_DIV\_34
  - LL\_RCC\_PLLM\_DIV\_35
  - LL\_RCC\_PLLM\_DIV\_36
  - LL\_RCC\_PLLM\_DIV\_37
  - LL\_RCC\_PLLM\_DIV\_38
  - LL\_RCC\_PLLM\_DIV\_39
  - LL\_RCC\_PLLM\_DIV\_40
  - LL\_RCC\_PLLM\_DIV\_41
  - LL\_RCC\_PLLM\_DIV\_42
  - LL\_RCC\_PLLM\_DIV\_43
  - LL\_RCC\_PLLM\_DIV\_44
  - LL\_RCC\_PLLM\_DIV\_45
  - LL\_RCC\_PLLM\_DIV\_46
  - LL\_RCC\_PLLM\_DIV\_47
  - LL\_RCC\_PLLM\_DIV\_48
  - LL\_RCC\_PLLM\_DIV\_49
  - LL\_RCC\_PLLM\_DIV\_50
  - LL\_RCC\_PLLM\_DIV\_51
  - LL\_RCC\_PLLM\_DIV\_52
  - LL\_RCC\_PLLM\_DIV\_53

- **PLLN:** Between 50 and 432
- **PLL2R:** This parameter can be one of the following values:
  - LL\_RCC\_PLLR\_DIV\_2
  - LL\_RCC\_PLLR\_DIV\_3
  - LL\_RCC\_PLLR\_DIV\_4
  - LL\_RCC\_PLLR\_DIV\_5
  - LL\_RCC\_PLLR\_DIV\_6
  - LL\_RCC\_PLLR\_DIV\_7
- **PLLDIVR:** This parameter can be one of the following values:
  - LL\_RCC\_PLLDIVR\_DIV\_1 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_2 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_3 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_4 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_5 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_6 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_7 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_8 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_9 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_10 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_11 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_12 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_13 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_14 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_15 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_16 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_17 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_18 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_19 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_20 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_21 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_22 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_23 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_24 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_25 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_26 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_27 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_28 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_29 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_30 (\*)
  - LL\_RCC\_PLLDIVR\_DIV\_31 (\*)

(\*) value not defined in all devices.

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI are disabled
- PLLN/PLL2R can be written only when PLL is disabled
- This can be selected for SAI

**Reference Manual to LL API cross reference:**

- PLLCFGR PLLSRC LL\_RCC\_PLL\_ConfigDomain\_SAI
- PLLCFGR PLLM LL\_RCC\_PLL\_ConfigDomain\_SAI
- PLLCFGR PLLN LL\_RCC\_PLL\_ConfigDomain\_SAI
- PLLCFGR PLLR LL\_RCC\_PLL\_ConfigDomain\_SAI
- DCKCFGR PLLDIVR LL\_RCC\_PLL\_ConfigDomain\_SAI

**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\_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\_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:** 50/192(\*) and 432

**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\_4
  - LL\_RCC\_PLLP\_DIV\_6
  - LL\_RCC\_PLLP\_DIV\_8

#### Reference Manual to LL API cross reference:

- 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\_3
  - LL\_RCC\_PLLQ\_DIV\_4
  - LL\_RCC\_PLLQ\_DIV\_5
  - LL\_RCC\_PLLQ\_DIV\_6
  - LL\_RCC\_PLLQ\_DIV\_7
  - LL\_RCC\_PLLQ\_DIV\_8
  - LL\_RCC\_PLLQ\_DIV\_9
  - LL\_RCC\_PLLQ\_DIV\_10
  - LL\_RCC\_PLLQ\_DIV\_11
  - LL\_RCC\_PLLQ\_DIV\_12
  - LL\_RCC\_PLLQ\_DIV\_13
  - LL\_RCC\_PLLQ\_DIV\_14
  - LL\_RCC\_PLLQ\_DIV\_15

#### Notes

- used for PLL48MCLK selected for USB, RNG, SDIO (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\_3
  - LL\_RCC\_PLLR\_DIV\_4
  - LL\_RCC\_PLLR\_DIV\_5
  - LL\_RCC\_PLLR\_DIV\_6
  - LL\_RCC\_PLLR\_DIV\_7

### 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\_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
- LL\_RCC\_PLLM\_DIV\_17
- LL\_RCC\_PLLM\_DIV\_18
- LL\_RCC\_PLLM\_DIV\_19
- LL\_RCC\_PLLM\_DIV\_20
- LL\_RCC\_PLLM\_DIV\_21
- LL\_RCC\_PLLM\_DIV\_22
- LL\_RCC\_PLLM\_DIV\_23
- LL\_RCC\_PLLM\_DIV\_24
- LL\_RCC\_PLLM\_DIV\_25
- LL\_RCC\_PLLM\_DIV\_26
- LL\_RCC\_PLLM\_DIV\_27
- LL\_RCC\_PLLM\_DIV\_28
- LL\_RCC\_PLLM\_DIV\_29
- LL\_RCC\_PLLM\_DIV\_30
- LL\_RCC\_PLLM\_DIV\_31
- LL\_RCC\_PLLM\_DIV\_32
- LL\_RCC\_PLLM\_DIV\_33
- LL\_RCC\_PLLM\_DIV\_34
- LL\_RCC\_PLLM\_DIV\_35
- LL\_RCC\_PLLM\_DIV\_36
- LL\_RCC\_PLLM\_DIV\_37
- LL\_RCC\_PLLM\_DIV\_38
- LL\_RCC\_PLLM\_DIV\_39
- LL\_RCC\_PLLM\_DIV\_40
- LL\_RCC\_PLLM\_DIV\_41
- LL\_RCC\_PLLM\_DIV\_42
- LL\_RCC\_PLLM\_DIV\_43
- LL\_RCC\_PLLM\_DIV\_44
- LL\_RCC\_PLLM\_DIV\_45
- LL\_RCC\_PLLM\_DIV\_46
- LL\_RCC\_PLLM\_DIV\_47
- LL\_RCC\_PLLM\_DIV\_48
- LL\_RCC\_PLLM\_DIV\_49
- LL\_RCC\_PLLM\_DIV\_50
- LL\_RCC\_PLLM\_DIV\_51
- LL\_RCC\_PLLM\_DIV\_52

**Reference Manual to LL API cross reference:**

- PLLCFGR PLLM LL\_RCC\_PLL\_GetDivider

**LL\_RCC\_PLL\_ConfigSpreadSpectrum**

**Function name**

`__STATIC_INLINE void LL_RCC_PLL_ConfigSpreadSpectrum (uint32_t Mod, uint32_t Inc, uint32_t Sel)`

**Function description**

Configure Spread Spectrum used for PLL.

**Parameters**

- **Mod:** Between Min\_Data=0 and Max\_Data=8191
- **Inc:** Between Min\_Data=0 and Max\_Data=32767
- **Sel:** This parameter can be one of the following values:
  - LL\_RCC\_SPREAD\_SELECT\_CENTER
  - LL\_RCC\_SPREAD\_SELECT\_DOWN

**Return values**

- **None:**

**Notes**

- These bits must be written before enabling PLL

**Reference Manual to LL API cross reference:**

- SSCGR MODPER LL\_RCC\_PLL\_ConfigSpreadSpectrum
- SSCGR INCSTEP LL\_RCC\_PLL\_ConfigSpreadSpectrum
- SSCGR SPREADSEL LL\_RCC\_PLL\_ConfigSpreadSpectrum

**LL\_RCC\_PLL\_GetPeriodModulation**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLL_GetPeriodModulation (void )`

**Function description**

Get Spread Spectrum Modulation Period for PLL.

**Return values**

- **Between:** Min\_Data=0 and Max\_Data=8191

**Reference Manual to LL API cross reference:**

- SSCGR MODPER LL\_RCC\_PLL\_GetPeriodModulation

**LL\_RCC\_PLL\_GetStepIncrementation**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLL_GetStepIncrementation (void )`

**Function description**

Get Spread Spectrum Incrementation Step for PLL.

**Return values**

- **Between:** Min\_Data=0 and Max\_Data=32767

**Notes**

- Must be written before enabling PLL

**Reference Manual to LL API cross reference:**

- SSCGR INCSTEP LL\_RCC\_PLL\_GetStepIncrementation

**LL\_RCC\_PLL\_GetSpreadSelection**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLL_GetSpreadSelection (void )`

**Function description**

Get Spread Spectrum Selection for PLL.

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_RCC\_SPREAD\_SELECT\_CENTER
  - LL\_RCC\_SPREAD\_SELECT\_DOWN

**Notes**

- Must be written before enabling PLL

**Reference Manual to LL API cross reference:**

- SSCGR SPREADSEL LL\_RCC\_PLL\_GetSpreadSelection

**LL\_RCC\_PLL\_SpreadSpectrum\_Enable**

**Function name**

`__STATIC_INLINE void LL_RCC_PLL_SpreadSpectrum_Enable (void )`

**Function description**

Enable Spread Spectrum for PLL.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- SSCGR SSCGEN LL\_RCC\_PLL\_SpreadSpectrum\_Enable

**LL\_RCC\_PLL\_SpreadSpectrum\_Disable**

**Function name**

`__STATIC_INLINE void LL_RCC_PLL_SpreadSpectrum_Disable (void )`

**Function description**

Disable Spread Spectrum for PLL.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- SSCGR SSCGEN LL\_RCC\_PLL\_SpreadSpectrum\_Disable

**LL\_RCC\_PLLI2S\_Enable**

**Function name**

`__STATIC_INLINE void LL_RCC_PLLI2S_Enable (void )`

**Function description**

Enable PLLI2S.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR PLLI2SON LL\_RCC\_PLLI2S\_Enable

**LL\_RCC\_PLLI2S\_Disable**

**Function name**

**\_\_STATIC\_INLINE void LL\_RCC\_PLLI2S\_Disable (void )**

**Function description**

Disable PLLI2S.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR PLLI2SON LL\_RCC\_PLLI2S\_Disable

**LL\_RCC\_PLLI2S\_IsReady**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_RCC\_PLLI2S\_IsReady (void )**

**Function description**

Check if PLLI2S Ready.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- CR PLLI2SRDY LL\_RCC\_PLLI2S\_IsReady

**LL\_RCC\_PLLI2S\_ConfigDomain\_SAI**

**Function name**

**\_\_STATIC\_INLINE void LL\_RCC\_PLLI2S\_ConfigDomain\_SAI (uint32\_t Source, uint32\_t PLLM, uint32\_t PLLN, uint32\_t PLLQ\_R, uint32\_t PLLDIVQ\_R)**

**Function description**

Configure PLLI2S used for SAI domain clock.

## Parameters

- **Source:** This parameter can be one of the following values:
    - LL\_RCC\_PLLSOURCE\_HSI
    - LL\_RCC\_PLLSOURCE\_HSE
    - LL\_RCC\_PLLI2SSOURCE\_PIN (\*)
- (\*) value not defined in all devices.

- **PLLM:** This parameter can be one of the following values:
  - LL\_RCC\_PLLI2SM\_DIV\_2
  - LL\_RCC\_PLLI2SM\_DIV\_3
  - LL\_RCC\_PLLI2SM\_DIV\_4
  - LL\_RCC\_PLLI2SM\_DIV\_5
  - LL\_RCC\_PLLI2SM\_DIV\_6
  - LL\_RCC\_PLLI2SM\_DIV\_7
  - LL\_RCC\_PLLI2SM\_DIV\_8
  - LL\_RCC\_PLLI2SM\_DIV\_9
  - LL\_RCC\_PLLI2SM\_DIV\_10
  - LL\_RCC\_PLLI2SM\_DIV\_11
  - LL\_RCC\_PLLI2SM\_DIV\_12
  - LL\_RCC\_PLLI2SM\_DIV\_13
  - LL\_RCC\_PLLI2SM\_DIV\_14
  - LL\_RCC\_PLLI2SM\_DIV\_15
  - LL\_RCC\_PLLI2SM\_DIV\_16
  - LL\_RCC\_PLLI2SM\_DIV\_17
  - LL\_RCC\_PLLI2SM\_DIV\_18
  - LL\_RCC\_PLLI2SM\_DIV\_19
  - LL\_RCC\_PLLI2SM\_DIV\_20
  - LL\_RCC\_PLLI2SM\_DIV\_21
  - LL\_RCC\_PLLI2SM\_DIV\_22
  - LL\_RCC\_PLLI2SM\_DIV\_23
  - LL\_RCC\_PLLI2SM\_DIV\_24
  - LL\_RCC\_PLLI2SM\_DIV\_25
  - LL\_RCC\_PLLI2SM\_DIV\_26
  - LL\_RCC\_PLLI2SM\_DIV\_27
  - LL\_RCC\_PLLI2SM\_DIV\_28
  - LL\_RCC\_PLLI2SM\_DIV\_29
  - LL\_RCC\_PLLI2SM\_DIV\_30
  - LL\_RCC\_PLLI2SM\_DIV\_31
  - LL\_RCC\_PLLI2SM\_DIV\_32
  - LL\_RCC\_PLLI2SM\_DIV\_33
  - LL\_RCC\_PLLI2SM\_DIV\_34
  - LL\_RCC\_PLLI2SM\_DIV\_35
  - LL\_RCC\_PLLI2SM\_DIV\_36
  - LL\_RCC\_PLLI2SM\_DIV\_37
  - LL\_RCC\_PLLI2SM\_DIV\_38
  - LL\_RCC\_PLLI2SM\_DIV\_39
  - LL\_RCC\_PLLI2SM\_DIV\_40
  - LL\_RCC\_PLLI2SM\_DIV\_41
  - LL\_RCC\_PLLI2SM\_DIV\_42
  - LL\_RCC\_PLLI2SM\_DIV\_43
  - LL\_RCC\_PLLI2SM\_DIV\_44
  - LL\_RCC\_PLLI2SM\_DIV\_45
  - LL\_RCC\_PLLI2SM\_DIV\_46
  - LL\_RCC\_PLLI2SM\_DIV\_47
  - LL\_RCC\_PLLI2SM\_DIV\_48
  - LL\_RCC\_PLLI2SM\_DIV\_49
  - LL\_RCC\_PLLI2SM\_DIV\_50
  - LL\_RCC\_PLLI2SM\_DIV\_51
  - LL\_RCC\_PLLI2SM\_DIV\_52
  - LL\_RCC\_PLLI2SM\_DIV\_53

- **PLLN:** Between 50/192(\*) and 432
- **PLLQ\_R:** This parameter can be one of the following values:
  - LL\_RCC\_PLLI2SQ\_DIV\_2 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_3 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_4 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_5 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_6 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_7 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_8 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_9 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_10 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_11 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_12 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_13 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_14 (\*)
  - LL\_RCC\_PLLI2SQ\_DIV\_15 (\*)
  - LL\_RCC\_PLLI2SR\_DIV\_2 (\*)
  - LL\_RCC\_PLLI2SR\_DIV\_3 (\*)
  - LL\_RCC\_PLLI2SR\_DIV\_4 (\*)
  - LL\_RCC\_PLLI2SR\_DIV\_5 (\*)
  - LL\_RCC\_PLLI2SR\_DIV\_6 (\*)
  - LL\_RCC\_PLLI2SR\_DIV\_7 (\*)

(\*) value not defined in all devices.



- **PLLDIVQ\_R:** This parameter can be one of the following values:
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_1 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_2 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_3 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_4 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_5 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_6 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_7 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_8 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_9 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_10 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_11 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_12 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_13 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_14 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_15 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_16 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_17 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_18 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_19 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_20 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_21 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_22 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_23 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_24 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_25 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_26 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_27 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_28 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_29 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_30 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_31 (\*)
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_32 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_1 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_2 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_3 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_4 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_5 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_6 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_7 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_8 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_9 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_10 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_11 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_12 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_13 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_14 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_15 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_16 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_17 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_18 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_19 (\*)
  - LL\_RCC\_PLLI2SDIVR\_DIV\_20 (\*)

### Return values

- **None:**

### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLLQ/PLLR can be written only when PLLI2S is disabled
- This can be selected for SAI

### Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- PLLI2SCFGR PLLI2SSRC LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- PLLCFGR PLLM LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- PLLI2SCFGR PLLI2SM LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- PLLI2SCFGR PLLI2SN LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- PLLI2SCFGR PLLI2SQ LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- PLLI2SCFGR PLLI2SR LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- DCKCFGR PLLI2SDIVQ LL\_RCC\_PLLI2S\_ConfigDomain\_SAI
- DCKCFGR PLLI2SDIVR LL\_RCC\_PLLI2S\_ConfigDomain\_SAI

### **LL\_RCC\_PLLI2S\_ConfigDomain\_I2S**

#### Function name

```
__STATIC_INLINE void LL_RCC_PLLI2S_ConfigDomain_I2S (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)
```

#### Function description

Configure PLLI2S used for I2S1 domain clock.

## Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE
  - LL\_RCC\_PLLI2SSOURCE\_PIN (\*)(\*) value not defined in all devices.

- **PLLM:** This parameter can be one of the following values:
  - LL\_RCC\_PLLI2SM\_DIV\_2
  - LL\_RCC\_PLLI2SM\_DIV\_3
  - LL\_RCC\_PLLI2SM\_DIV\_4
  - LL\_RCC\_PLLI2SM\_DIV\_5
  - LL\_RCC\_PLLI2SM\_DIV\_6
  - LL\_RCC\_PLLI2SM\_DIV\_7
  - LL\_RCC\_PLLI2SM\_DIV\_8
  - LL\_RCC\_PLLI2SM\_DIV\_9
  - LL\_RCC\_PLLI2SM\_DIV\_10
  - LL\_RCC\_PLLI2SM\_DIV\_11
  - LL\_RCC\_PLLI2SM\_DIV\_12
  - LL\_RCC\_PLLI2SM\_DIV\_13
  - LL\_RCC\_PLLI2SM\_DIV\_14
  - LL\_RCC\_PLLI2SM\_DIV\_15
  - LL\_RCC\_PLLI2SM\_DIV\_16
  - LL\_RCC\_PLLI2SM\_DIV\_17
  - LL\_RCC\_PLLI2SM\_DIV\_18
  - LL\_RCC\_PLLI2SM\_DIV\_19
  - LL\_RCC\_PLLI2SM\_DIV\_20
  - LL\_RCC\_PLLI2SM\_DIV\_21
  - LL\_RCC\_PLLI2SM\_DIV\_22
  - LL\_RCC\_PLLI2SM\_DIV\_23
  - LL\_RCC\_PLLI2SM\_DIV\_24
  - LL\_RCC\_PLLI2SM\_DIV\_25
  - LL\_RCC\_PLLI2SM\_DIV\_26
  - LL\_RCC\_PLLI2SM\_DIV\_27
  - LL\_RCC\_PLLI2SM\_DIV\_28
  - LL\_RCC\_PLLI2SM\_DIV\_29
  - LL\_RCC\_PLLI2SM\_DIV\_30
  - LL\_RCC\_PLLI2SM\_DIV\_31
  - LL\_RCC\_PLLI2SM\_DIV\_32
  - LL\_RCC\_PLLI2SM\_DIV\_33
  - LL\_RCC\_PLLI2SM\_DIV\_34
  - LL\_RCC\_PLLI2SM\_DIV\_35
  - LL\_RCC\_PLLI2SM\_DIV\_36
  - LL\_RCC\_PLLI2SM\_DIV\_37
  - LL\_RCC\_PLLI2SM\_DIV\_38
  - LL\_RCC\_PLLI2SM\_DIV\_39
  - LL\_RCC\_PLLI2SM\_DIV\_40
  - LL\_RCC\_PLLI2SM\_DIV\_41
  - LL\_RCC\_PLLI2SM\_DIV\_42
  - LL\_RCC\_PLLI2SM\_DIV\_43
  - LL\_RCC\_PLLI2SM\_DIV\_44
  - LL\_RCC\_PLLI2SM\_DIV\_45
  - LL\_RCC\_PLLI2SM\_DIV\_46
  - LL\_RCC\_PLLI2SM\_DIV\_47
  - LL\_RCC\_PLLI2SM\_DIV\_48
  - LL\_RCC\_PLLI2SM\_DIV\_49
  - LL\_RCC\_PLLI2SM\_DIV\_50
  - LL\_RCC\_PLLI2SM\_DIV\_51
  - LL\_RCC\_PLLI2SM\_DIV\_52
  - LL\_RCC\_PLLI2SM\_DIV\_53

- **PLLN:** Between 50/192(\*) and 432
- **PLLR:** This parameter can be one of the following values:
  - LL\_RCC\_PLLI2SR\_DIV\_2
  - LL\_RCC\_PLLI2SR\_DIV\_3
  - LL\_RCC\_PLLI2SR\_DIV\_4
  - LL\_RCC\_PLLI2SR\_DIV\_5
  - LL\_RCC\_PLLI2SR\_DIV\_6
  - LL\_RCC\_PLLI2SR\_DIV\_7

**Return values**

- **None:**

**Notes**

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLLR can be written only when PLLI2S is disabled
- This can be selected for I2S

**Reference Manual to LL API cross reference:**

- PLLCFGR PLLSRC LL\_RCC\_PLLI2S\_ConfigDomain\_I2S
- PLLCFGR PLLM LL\_RCC\_PLLI2S\_ConfigDomain\_I2S
- PLLI2SCFGR PLLI2SSRC LL\_RCC\_PLLI2S\_ConfigDomain\_I2S
- PLLI2SCFGR PLLI2SM LL\_RCC\_PLLI2S\_ConfigDomain\_I2S
- PLLI2SCFGR PLLI2SN LL\_RCC\_PLLI2S\_ConfigDomain\_I2S
- PLLI2SCFGR PLLI2SR LL\_RCC\_PLLI2S\_ConfigDomain\_I2S

**LL\_RCC\_PLLI2S\_GetN**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLI2S_GetN (void )`

**Function description**

Get I2SPLL multiplication factor for VCO.

**Return values**

- **Between:** 50/192(\*) and 432

**Reference Manual to LL API cross reference:**

- PLLI2SCFGR PLLI2SN LL\_RCC\_PLLI2S\_GetN

**LL\_RCC\_PLLI2S\_GetQ**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLI2S_GetQ (void )`

**Function description**

Get I2SPLL division factor for PLLI2SQ.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLI2SQ\_DIV\_2
  - LL\_RCC\_PLLI2SQ\_DIV\_3
  - LL\_RCC\_PLLI2SQ\_DIV\_4
  - LL\_RCC\_PLLI2SQ\_DIV\_5
  - LL\_RCC\_PLLI2SQ\_DIV\_6
  - LL\_RCC\_PLLI2SQ\_DIV\_7
  - LL\_RCC\_PLLI2SQ\_DIV\_8
  - LL\_RCC\_PLLI2SQ\_DIV\_9
  - LL\_RCC\_PLLI2SQ\_DIV\_10
  - LL\_RCC\_PLLI2SQ\_DIV\_11
  - LL\_RCC\_PLLI2SQ\_DIV\_12
  - LL\_RCC\_PLLI2SQ\_DIV\_13
  - LL\_RCC\_PLLI2SQ\_DIV\_14
  - LL\_RCC\_PLLI2SQ\_DIV\_15

### Reference Manual to LL API cross reference:

- PLLI2SCFGR PLLI2SQ LL\_RCC\_PLLI2S\_GetQ

#### LL\_RCC\_PLLI2S\_GetR

### Function name

`__STATIC_INLINE uint32_t LL_RCC_PLLI2S_GetR (void )`

### Function description

Get I2SPLL division factor for PLLI2SR.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLI2SR\_DIV\_2
  - LL\_RCC\_PLLI2SR\_DIV\_3
  - LL\_RCC\_PLLI2SR\_DIV\_4
  - LL\_RCC\_PLLI2SR\_DIV\_5
  - LL\_RCC\_PLLI2SR\_DIV\_6
  - LL\_RCC\_PLLI2SR\_DIV\_7

### Notes

- used for PLLI2SCLK (I2S clock)

### Reference Manual to LL API cross reference:

- PLLI2SCFGR PLLI2SR LL\_RCC\_PLLI2S\_GetR

#### LL\_RCC\_PLLI2S\_GetDIVQ

### Function name

`__STATIC_INLINE uint32_t LL_RCC_PLLI2S_GetDIVQ (void )`

### Function description

Get I2SPLL division factor for PLLI2SDIVQ.

## Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_1
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_2
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_3
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_4
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_5
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_6
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_7
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_8
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_9
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_10
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_11
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_12
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_13
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_14
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_15
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_16
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_17
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_18
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_19
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_20
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_21
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_22
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_23
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_24
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_25
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_26
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_27
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_28
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_29
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_30
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_31
  - LL\_RCC\_PLLI2SDIVQ\_DIV\_32

## Notes

- used PLLSAICLK selected (SAI clock)

## Reference Manual to LL API cross reference:

- DCKCFGR PLLI2SDIVQ LL\_RCC\_PLLI2S\_GetDIVQ

### LL\_RCC\_PLLI2S\_GetDivider

## Function name

`__STATIC_INLINE uint32_t LL_RCC_PLLI2S_GetDivider (void )`

## Function description

Get division factor for PLLI2S input clock.

**Return values**

- **Returned:** value can be one of the following values:

- LL\_RCC\_PLLI2SM\_DIV\_2
- LL\_RCC\_PLLI2SM\_DIV\_3
- LL\_RCC\_PLLI2SM\_DIV\_4
- LL\_RCC\_PLLI2SM\_DIV\_5
- LL\_RCC\_PLLI2SM\_DIV\_6
- LL\_RCC\_PLLI2SM\_DIV\_7
- LL\_RCC\_PLLI2SM\_DIV\_8
- LL\_RCC\_PLLI2SM\_DIV\_9
- LL\_RCC\_PLLI2SM\_DIV\_10
- LL\_RCC\_PLLI2SM\_DIV\_11
- LL\_RCC\_PLLI2SM\_DIV\_12
- LL\_RCC\_PLLI2SM\_DIV\_13
- LL\_RCC\_PLLI2SM\_DIV\_14
- LL\_RCC\_PLLI2SM\_DIV\_15
- LL\_RCC\_PLLI2SM\_DIV\_16
- LL\_RCC\_PLLI2SM\_DIV\_17
- LL\_RCC\_PLLI2SM\_DIV\_18
- LL\_RCC\_PLLI2SM\_DIV\_19
- LL\_RCC\_PLLI2SM\_DIV\_20
- LL\_RCC\_PLLI2SM\_DIV\_21
- LL\_RCC\_PLLI2SM\_DIV\_22
- LL\_RCC\_PLLI2SM\_DIV\_23
- LL\_RCC\_PLLI2SM\_DIV\_24
- LL\_RCC\_PLLI2SM\_DIV\_25
- LL\_RCC\_PLLI2SM\_DIV\_26
- LL\_RCC\_PLLI2SM\_DIV\_27
- LL\_RCC\_PLLI2SM\_DIV\_28
- LL\_RCC\_PLLI2SM\_DIV\_29
- LL\_RCC\_PLLI2SM\_DIV\_30
- LL\_RCC\_PLLI2SM\_DIV\_31
- LL\_RCC\_PLLI2SM\_DIV\_32
- LL\_RCC\_PLLI2SM\_DIV\_33
- LL\_RCC\_PLLI2SM\_DIV\_34
- LL\_RCC\_PLLI2SM\_DIV\_35
- LL\_RCC\_PLLI2SM\_DIV\_36
- LL\_RCC\_PLLI2SM\_DIV\_37
- LL\_RCC\_PLLI2SM\_DIV\_38
- LL\_RCC\_PLLI2SM\_DIV\_39
- LL\_RCC\_PLLI2SM\_DIV\_40
- LL\_RCC\_PLLI2SM\_DIV\_41
- LL\_RCC\_PLLI2SM\_DIV\_42
- LL\_RCC\_PLLI2SM\_DIV\_43
- LL\_RCC\_PLLI2SM\_DIV\_44
- LL\_RCC\_PLLI2SM\_DIV\_45
- LL\_RCC\_PLLI2SM\_DIV\_46
- LL\_RCC\_PLLI2SM\_DIV\_47
- LL\_RCC\_PLLI2SM\_DIV\_48
- LL\_RCC\_PLLI2SM\_DIV\_49
- LL\_RCC\_PLLI2SM\_DIV\_50
- LL\_RCC\_PLLI2SM\_DIV\_51
- LL\_RCC\_PLLI2SM\_DIV\_52



**Reference Manual to LL API cross reference:**

- PLLCFGR PLLM LL\_RCC\_PLLI2S\_GetDivider
- PLLI2SCFGR PLLI2SM LL\_RCC\_PLLI2S\_GetDivider

**LL\_RCC\_PLLI2S\_GetMainSource**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLI2S_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\_HSI
    - LL\_RCC\_PLLSOURCE\_HSE
    - LL\_RCC\_PLLI2SSOURCE\_PIN (\*)
- (\*) value not defined in all devices.

**Reference Manual to LL API cross reference:**

- PLLCFGR PLLSRC LL\_RCC\_PLLI2S\_GetMainSource
- PLLI2SCFGR PLLI2SSRC LL\_RCC\_PLLI2S\_GetMainSource

**LL\_RCC\_PLLSAI\_Enable**

**Function name**

`__STATIC_INLINE void LL_RCC_PLLSAI_Enable (void )`

**Function description**

Enable PLLSAI.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR PLLSAION LL\_RCC\_PLLSAI\_Enable

**LL\_RCC\_PLLSAI\_Disable**

**Function name**

`__STATIC_INLINE void LL_RCC_PLLSAI_Disable (void )`

**Function description**

Disable PLLSAI.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR PLLSAION LL\_RCC\_PLLSAI\_Disable

**LL\_RCC\_PLLSAI\_IsReady**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLSAI_IsReady (void )`

### Function description

Check if PLLSAI Ready.

### Return values

- **State:** of bit (1 or 0).

### Reference Manual to LL API cross reference:

- CR PLLSAIRDY LL\_RCC\_PLLSAI\_IsReady

### LL\_RCC\_PLLSAI\_ConfigDomain\_SAI

### Function name

```
__STATIC_INLINE void LL_RCC_PLLSAI_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t  
PLLN, uint32_t PLLQ, uint32_t PLLDIVQ)
```

### Function description

Configure PLLSAI used for SAI domain clock.

### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLL\_M:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIM\_DIV\_2
  - LL\_RCC\_PLLSAIM\_DIV\_3
  - LL\_RCC\_PLLSAIM\_DIV\_4
  - LL\_RCC\_PLLSAIM\_DIV\_5
  - LL\_RCC\_PLLSAIM\_DIV\_6
  - LL\_RCC\_PLLSAIM\_DIV\_7
  - LL\_RCC\_PLLSAIM\_DIV\_8
  - LL\_RCC\_PLLSAIM\_DIV\_9
  - LL\_RCC\_PLLSAIM\_DIV\_10
  - LL\_RCC\_PLLSAIM\_DIV\_11
  - LL\_RCC\_PLLSAIM\_DIV\_12
  - LL\_RCC\_PLLSAIM\_DIV\_13
  - LL\_RCC\_PLLSAIM\_DIV\_14
  - LL\_RCC\_PLLSAIM\_DIV\_15
  - LL\_RCC\_PLLSAIM\_DIV\_16
  - LL\_RCC\_PLLSAIM\_DIV\_17
  - LL\_RCC\_PLLSAIM\_DIV\_18
  - LL\_RCC\_PLLSAIM\_DIV\_19
  - LL\_RCC\_PLLSAIM\_DIV\_20
  - LL\_RCC\_PLLSAIM\_DIV\_21
  - LL\_RCC\_PLLSAIM\_DIV\_22
  - LL\_RCC\_PLLSAIM\_DIV\_23
  - LL\_RCC\_PLLSAIM\_DIV\_24
  - LL\_RCC\_PLLSAIM\_DIV\_25
  - LL\_RCC\_PLLSAIM\_DIV\_26
  - LL\_RCC\_PLLSAIM\_DIV\_27
  - LL\_RCC\_PLLSAIM\_DIV\_28
  - LL\_RCC\_PLLSAIM\_DIV\_29
  - LL\_RCC\_PLLSAIM\_DIV\_30
  - LL\_RCC\_PLLSAIM\_DIV\_31
  - LL\_RCC\_PLLSAIM\_DIV\_32
  - LL\_RCC\_PLLSAIM\_DIV\_33
  - LL\_RCC\_PLLSAIM\_DIV\_34
  - LL\_RCC\_PLLSAIM\_DIV\_35
  - LL\_RCC\_PLLSAIM\_DIV\_36
  - LL\_RCC\_PLLSAIM\_DIV\_37
  - LL\_RCC\_PLLSAIM\_DIV\_38
  - LL\_RCC\_PLLSAIM\_DIV\_39
  - LL\_RCC\_PLLSAIM\_DIV\_40
  - LL\_RCC\_PLLSAIM\_DIV\_41
  - LL\_RCC\_PLLSAIM\_DIV\_42
  - LL\_RCC\_PLLSAIM\_DIV\_43
  - LL\_RCC\_PLLSAIM\_DIV\_44
  - LL\_RCC\_PLLSAIM\_DIV\_45
  - LL\_RCC\_PLLSAIM\_DIV\_46
  - LL\_RCC\_PLLSAIM\_DIV\_47
  - LL\_RCC\_PLLSAIM\_DIV\_48
  - LL\_RCC\_PLLSAIM\_DIV\_49
  - LL\_RCC\_PLLSAIM\_DIV\_50
  - LL\_RCC\_PLLSAIM\_DIV\_51
  - LL\_RCC\_PLLSAIM\_DIV\_52
  - LL\_RCC\_PLLSAIM\_DIV\_53

- **PLLN:** Between 49/50(\*) and 432
- **PLLQ:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIQ\_DIV\_2
  - LL\_RCC\_PLLSAIQ\_DIV\_3
  - LL\_RCC\_PLLSAIQ\_DIV\_4
  - LL\_RCC\_PLLSAIQ\_DIV\_5
  - LL\_RCC\_PLLSAIQ\_DIV\_6
  - LL\_RCC\_PLLSAIQ\_DIV\_7
  - LL\_RCC\_PLLSAIQ\_DIV\_8
  - LL\_RCC\_PLLSAIQ\_DIV\_9
  - LL\_RCC\_PLLSAIQ\_DIV\_10
  - LL\_RCC\_PLLSAIQ\_DIV\_11
  - LL\_RCC\_PLLSAIQ\_DIV\_12
  - LL\_RCC\_PLLSAIQ\_DIV\_13
  - LL\_RCC\_PLLSAIQ\_DIV\_14
  - LL\_RCC\_PLLSAIQ\_DIV\_15
- **PLLDIVQ:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_1
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_2
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_3
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_4
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_5
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_6
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_7
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_8
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_9
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_10
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_11
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_12
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_13
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_14
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_15
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_16
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_17
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_18
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_19
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_20
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_21
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_22
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_23
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_24
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_25
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_26
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_27
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_28
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_29
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_30
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_31
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_32

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLLQ can be written only when PLLSAI is disabled
- This can be selected for SAI

#### Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL\_RCC\_PLLSAI\_ConfigDomain\_SAI
- PLLCFGR PLLM LL\_RCC\_PLLSAI\_ConfigDomain\_SAI
- PLLSAICFGR PLLSAIM LL\_RCC\_PLLSAI\_ConfigDomain\_SAI
- PLLSAICFGR PLLSAIN LL\_RCC\_PLLSAI\_ConfigDomain\_SAI
- PLLSAICFGR PLLSAIQ LL\_RCC\_PLLSAI\_ConfigDomain\_SAI
- DCKCFGR PLLSAIDIVQ LL\_RCC\_PLLSAI\_ConfigDomain\_SAI

#### **LL\_RCC\_PLLSAI\_ConfigDomain\_48M**

#### Function name

```
__STATIC_INLINE void LL_RCC_PLLSAI_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)
```

#### Function description

Configure PLLSAI used for 48Mhz domain clock.

### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLL\_M:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIM\_DIV\_2
  - LL\_RCC\_PLLSAIM\_DIV\_3
  - LL\_RCC\_PLLSAIM\_DIV\_4
  - LL\_RCC\_PLLSAIM\_DIV\_5
  - LL\_RCC\_PLLSAIM\_DIV\_6
  - LL\_RCC\_PLLSAIM\_DIV\_7
  - LL\_RCC\_PLLSAIM\_DIV\_8
  - LL\_RCC\_PLLSAIM\_DIV\_9
  - LL\_RCC\_PLLSAIM\_DIV\_10
  - LL\_RCC\_PLLSAIM\_DIV\_11
  - LL\_RCC\_PLLSAIM\_DIV\_12
  - LL\_RCC\_PLLSAIM\_DIV\_13
  - LL\_RCC\_PLLSAIM\_DIV\_14
  - LL\_RCC\_PLLSAIM\_DIV\_15
  - LL\_RCC\_PLLSAIM\_DIV\_16
  - LL\_RCC\_PLLSAIM\_DIV\_17
  - LL\_RCC\_PLLSAIM\_DIV\_18
  - LL\_RCC\_PLLSAIM\_DIV\_19
  - LL\_RCC\_PLLSAIM\_DIV\_20
  - LL\_RCC\_PLLSAIM\_DIV\_21
  - LL\_RCC\_PLLSAIM\_DIV\_22
  - LL\_RCC\_PLLSAIM\_DIV\_23
  - LL\_RCC\_PLLSAIM\_DIV\_24
  - LL\_RCC\_PLLSAIM\_DIV\_25
  - LL\_RCC\_PLLSAIM\_DIV\_26
  - LL\_RCC\_PLLSAIM\_DIV\_27
  - LL\_RCC\_PLLSAIM\_DIV\_28
  - LL\_RCC\_PLLSAIM\_DIV\_29
  - LL\_RCC\_PLLSAIM\_DIV\_30
  - LL\_RCC\_PLLSAIM\_DIV\_31
  - LL\_RCC\_PLLSAIM\_DIV\_32
  - LL\_RCC\_PLLSAIM\_DIV\_33
  - LL\_RCC\_PLLSAIM\_DIV\_34
  - LL\_RCC\_PLLSAIM\_DIV\_35
  - LL\_RCC\_PLLSAIM\_DIV\_36
  - LL\_RCC\_PLLSAIM\_DIV\_37
  - LL\_RCC\_PLLSAIM\_DIV\_38
  - LL\_RCC\_PLLSAIM\_DIV\_39
  - LL\_RCC\_PLLSAIM\_DIV\_40
  - LL\_RCC\_PLLSAIM\_DIV\_41
  - LL\_RCC\_PLLSAIM\_DIV\_42
  - LL\_RCC\_PLLSAIM\_DIV\_43
  - LL\_RCC\_PLLSAIM\_DIV\_44
  - LL\_RCC\_PLLSAIM\_DIV\_45
  - LL\_RCC\_PLLSAIM\_DIV\_46
  - LL\_RCC\_PLLSAIM\_DIV\_47
  - LL\_RCC\_PLLSAIM\_DIV\_48
  - LL\_RCC\_PLLSAIM\_DIV\_49
  - LL\_RCC\_PLLSAIM\_DIV\_50
  - LL\_RCC\_PLLSAIM\_DIV\_51
  - LL\_RCC\_PLLSAIM\_DIV\_52
  - LL\_RCC\_PLLSAIM\_DIV\_53



- **PLLN:** Between 50 and 432
- **PLL P:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIP\_DIV\_2
  - LL\_RCC\_PLLSAIP\_DIV\_4
  - LL\_RCC\_PLLSAIP\_DIV\_6
  - LL\_RCC\_PLLSAIP\_DIV\_8

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLL P can be written only when PLLSAI is disabled
- This can be selected for USB, RNG, SDIO

#### Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL\_RCC\_PLLSAI\_ConfigDomain\_48M
- PLLCFGR PLLM LL\_RCC\_PLLSAI\_ConfigDomain\_48M
- PLLSAICFGR PLLSAIM LL\_RCC\_PLLSAI\_ConfigDomain\_48M
- PLLSAICFGR PLLSAIN LL\_RCC\_PLLSAI\_ConfigDomain\_48M
- PLLSAICFGR PLLSAIP LL\_RCC\_PLLSAI\_ConfigDomain\_48M

#### LL\_RCC\_PLLSAI\_ConfigDomain\_LTDC

#### Function name

```
__STATIC_INLINE void LL_RCC_PLLSAI_ConfigDomain_LTDC (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR, uint32_t PLLDIVR)
```

#### Function description

Configure PLLSAI used for LTDC domain clock.

### Parameters

- **Source:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSOURCE\_HSI
  - LL\_RCC\_PLLSOURCE\_HSE

- **PLL\_M:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIM\_DIV\_2
  - LL\_RCC\_PLLSAIM\_DIV\_3
  - LL\_RCC\_PLLSAIM\_DIV\_4
  - LL\_RCC\_PLLSAIM\_DIV\_5
  - LL\_RCC\_PLLSAIM\_DIV\_6
  - LL\_RCC\_PLLSAIM\_DIV\_7
  - LL\_RCC\_PLLSAIM\_DIV\_8
  - LL\_RCC\_PLLSAIM\_DIV\_9
  - LL\_RCC\_PLLSAIM\_DIV\_10
  - LL\_RCC\_PLLSAIM\_DIV\_11
  - LL\_RCC\_PLLSAIM\_DIV\_12
  - LL\_RCC\_PLLSAIM\_DIV\_13
  - LL\_RCC\_PLLSAIM\_DIV\_14
  - LL\_RCC\_PLLSAIM\_DIV\_15
  - LL\_RCC\_PLLSAIM\_DIV\_16
  - LL\_RCC\_PLLSAIM\_DIV\_17
  - LL\_RCC\_PLLSAIM\_DIV\_18
  - LL\_RCC\_PLLSAIM\_DIV\_19
  - LL\_RCC\_PLLSAIM\_DIV\_20
  - LL\_RCC\_PLLSAIM\_DIV\_21
  - LL\_RCC\_PLLSAIM\_DIV\_22
  - LL\_RCC\_PLLSAIM\_DIV\_23
  - LL\_RCC\_PLLSAIM\_DIV\_24
  - LL\_RCC\_PLLSAIM\_DIV\_25
  - LL\_RCC\_PLLSAIM\_DIV\_26
  - LL\_RCC\_PLLSAIM\_DIV\_27
  - LL\_RCC\_PLLSAIM\_DIV\_28
  - LL\_RCC\_PLLSAIM\_DIV\_29
  - LL\_RCC\_PLLSAIM\_DIV\_30
  - LL\_RCC\_PLLSAIM\_DIV\_31
  - LL\_RCC\_PLLSAIM\_DIV\_32
  - LL\_RCC\_PLLSAIM\_DIV\_33
  - LL\_RCC\_PLLSAIM\_DIV\_34
  - LL\_RCC\_PLLSAIM\_DIV\_35
  - LL\_RCC\_PLLSAIM\_DIV\_36
  - LL\_RCC\_PLLSAIM\_DIV\_37
  - LL\_RCC\_PLLSAIM\_DIV\_38
  - LL\_RCC\_PLLSAIM\_DIV\_39
  - LL\_RCC\_PLLSAIM\_DIV\_40
  - LL\_RCC\_PLLSAIM\_DIV\_41
  - LL\_RCC\_PLLSAIM\_DIV\_42
  - LL\_RCC\_PLLSAIM\_DIV\_43
  - LL\_RCC\_PLLSAIM\_DIV\_44
  - LL\_RCC\_PLLSAIM\_DIV\_45
  - LL\_RCC\_PLLSAIM\_DIV\_46
  - LL\_RCC\_PLLSAIM\_DIV\_47
  - LL\_RCC\_PLLSAIM\_DIV\_48
  - LL\_RCC\_PLLSAIM\_DIV\_49
  - LL\_RCC\_PLLSAIM\_DIV\_50
  - LL\_RCC\_PLLSAIM\_DIV\_51
  - LL\_RCC\_PLLSAIM\_DIV\_52
  - LL\_RCC\_PLLSAIM\_DIV\_53

- **PLLN:** Between 49/50(\*) and 432
- **PLLRR:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIR\_DIV\_2
  - LL\_RCC\_PLLSAIR\_DIV\_3
  - LL\_RCC\_PLLSAIR\_DIV\_4
  - LL\_RCC\_PLLSAIR\_DIV\_5
  - LL\_RCC\_PLLSAIR\_DIV\_6
  - LL\_RCC\_PLLSAIR\_DIV\_7
- **PLLDIVR:** This parameter can be one of the following values:
  - LL\_RCC\_PLLSAIDIVR\_DIV\_2
  - LL\_RCC\_PLLSAIDIVR\_DIV\_4
  - LL\_RCC\_PLLSAIDIVR\_DIV\_8
  - LL\_RCC\_PLLSAIDIVR\_DIV\_16

#### Return values

- **None:**

#### Notes

- PLL Source and PLLM Divider can be written only when PLL, PLLI2S and PLLSAI(\*) are disabled
- PLLN/PLLRR can be written only when PLLSAI is disabled
- This can be selected for LTDC

#### Reference Manual to LL API cross reference:

- PLLCFGR PLLSRC LL\_RCC\_PLLSAI\_ConfigDomain\_LTDC
- PLLCFGR PLLM LL\_RCC\_PLLSAI\_ConfigDomain\_LTDC
- PLLSAICFGR PLLSAIN LL\_RCC\_PLLSAI\_ConfigDomain\_LTDC
- PLLSAICFGR PLLSAIR LL\_RCC\_PLLSAI\_ConfigDomain\_LTDC
- DCKCFGR PLLSAIDIVR LL\_RCC\_PLLSAI\_ConfigDomain\_LTDC

#### **LL\_RCC\_PLLSAI\_GetDivider**

#### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_RCC\_PLLSAI\_GetDivider (void )**

#### Function description

Get division factor for PLLSAI input clock.

## Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLSAIM\_DIV\_2
  - LL\_RCC\_PLLSAIM\_DIV\_3
  - LL\_RCC\_PLLSAIM\_DIV\_4
  - LL\_RCC\_PLLSAIM\_DIV\_5
  - LL\_RCC\_PLLSAIM\_DIV\_6
  - LL\_RCC\_PLLSAIM\_DIV\_7
  - LL\_RCC\_PLLSAIM\_DIV\_8
  - LL\_RCC\_PLLSAIM\_DIV\_9
  - LL\_RCC\_PLLSAIM\_DIV\_10
  - LL\_RCC\_PLLSAIM\_DIV\_11
  - LL\_RCC\_PLLSAIM\_DIV\_12
  - LL\_RCC\_PLLSAIM\_DIV\_13
  - LL\_RCC\_PLLSAIM\_DIV\_14
  - LL\_RCC\_PLLSAIM\_DIV\_15
  - LL\_RCC\_PLLSAIM\_DIV\_16
  - LL\_RCC\_PLLSAIM\_DIV\_17
  - LL\_RCC\_PLLSAIM\_DIV\_18
  - LL\_RCC\_PLLSAIM\_DIV\_19
  - LL\_RCC\_PLLSAIM\_DIV\_20
  - LL\_RCC\_PLLSAIM\_DIV\_21
  - LL\_RCC\_PLLSAIM\_DIV\_22
  - LL\_RCC\_PLLSAIM\_DIV\_23
  - LL\_RCC\_PLLSAIM\_DIV\_24
  - LL\_RCC\_PLLSAIM\_DIV\_25
  - LL\_RCC\_PLLSAIM\_DIV\_26
  - LL\_RCC\_PLLSAIM\_DIV\_27
  - LL\_RCC\_PLLSAIM\_DIV\_28
  - LL\_RCC\_PLLSAIM\_DIV\_29
  - LL\_RCC\_PLLSAIM\_DIV\_30
  - LL\_RCC\_PLLSAIM\_DIV\_31
  - LL\_RCC\_PLLSAIM\_DIV\_32
  - LL\_RCC\_PLLSAIM\_DIV\_33
  - LL\_RCC\_PLLSAIM\_DIV\_34
  - LL\_RCC\_PLLSAIM\_DIV\_35
  - LL\_RCC\_PLLSAIM\_DIV\_36
  - LL\_RCC\_PLLSAIM\_DIV\_37
  - LL\_RCC\_PLLSAIM\_DIV\_38
  - LL\_RCC\_PLLSAIM\_DIV\_39
  - LL\_RCC\_PLLSAIM\_DIV\_40
  - LL\_RCC\_PLLSAIM\_DIV\_41
  - LL\_RCC\_PLLSAIM\_DIV\_42
  - LL\_RCC\_PLLSAIM\_DIV\_43
  - LL\_RCC\_PLLSAIM\_DIV\_44
  - LL\_RCC\_PLLSAIM\_DIV\_45
  - LL\_RCC\_PLLSAIM\_DIV\_46
  - LL\_RCC\_PLLSAIM\_DIV\_47
  - LL\_RCC\_PLLSAIM\_DIV\_48
  - LL\_RCC\_PLLSAIM\_DIV\_49
  - LL\_RCC\_PLLSAIM\_DIV\_50
  - LL\_RCC\_PLLSAIM\_DIV\_51
  - LL\_RCC\_PLLSAIM\_DIV\_52

**Reference Manual to LL API cross reference:**

- PLLCFGR PLLM LL\_RCC\_PLLSAI\_GetDivider
- PLLSAICFGR PLLSAIM LL\_RCC\_PLLSAI\_GetDivider

**LL\_RCC\_PLLSAI\_GetN**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLSAI_GetN (void )`

**Function description**

Get SAIPLL multiplication factor for VCO.

**Return values**

- **Between:** 49/50(\*) and 432

**Reference Manual to LL API cross reference:**

- PLLSAICFGR PLLSAIN LL\_RCC\_PLLSAI\_GetN

**LL\_RCC\_PLLSAI\_GetQ**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLSAI_GetQ (void )`

**Function description**

Get SAIPLL division factor for PLLSAIQ.

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLSAIQ\_DIV\_2
  - LL\_RCC\_PLLSAIQ\_DIV\_3
  - LL\_RCC\_PLLSAIQ\_DIV\_4
  - LL\_RCC\_PLLSAIQ\_DIV\_5
  - LL\_RCC\_PLLSAIQ\_DIV\_6
  - LL\_RCC\_PLLSAIQ\_DIV\_7
  - LL\_RCC\_PLLSAIQ\_DIV\_8
  - LL\_RCC\_PLLSAIQ\_DIV\_9
  - LL\_RCC\_PLLSAIQ\_DIV\_10
  - LL\_RCC\_PLLSAIQ\_DIV\_11
  - LL\_RCC\_PLLSAIQ\_DIV\_12
  - LL\_RCC\_PLLSAIQ\_DIV\_13
  - LL\_RCC\_PLLSAIQ\_DIV\_14
  - LL\_RCC\_PLLSAIQ\_DIV\_15

**Reference Manual to LL API cross reference:**

- PLLSAICFGR PLLSAIQ LL\_RCC\_PLLSAI\_GetQ

**LL\_RCC\_PLLSAI\_GetR**

**Function name**

`__STATIC_INLINE uint32_t LL_RCC_PLLSAI_GetR (void )`

**Function description**

Get SAIPLL division factor for PLLSAIR.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLSAIR\_DIV\_2
  - LL\_RCC\_PLLSAIR\_DIV\_3
  - LL\_RCC\_PLLSAIR\_DIV\_4
  - LL\_RCC\_PLLSAIR\_DIV\_5
  - LL\_RCC\_PLLSAIR\_DIV\_6
  - LL\_RCC\_PLLSAIR\_DIV\_7

### Notes

- used for PLLSAICLK (SAI clock)

### Reference Manual to LL API cross reference:

- PLLSAICFGR PLLSAIR LL\_RCC\_PLLSAI\_GetR

#### LL\_RCC\_PLLSAI\_GetP

### Function name

`__STATIC_INLINE uint32_t LL_RCC_PLLSAI_GetP (void )`

### Function description

Get SAIPLL division factor for PLLSAIP.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLSAIP\_DIV\_2
  - LL\_RCC\_PLLSAIP\_DIV\_4
  - LL\_RCC\_PLLSAIP\_DIV\_6
  - LL\_RCC\_PLLSAIP\_DIV\_8

### Notes

- used for PLL48MCLK (48M domain clock)

### Reference Manual to LL API cross reference:

- PLLSAICFGR PLLSAIP LL\_RCC\_PLLSAI\_GetP

#### LL\_RCC\_PLLSAI\_GetDIVQ

### Function name

`__STATIC_INLINE uint32_t LL_RCC_PLLSAI_GetDIVQ (void )`

### Function description

Get SAIPLL division factor for PLLSAIDIVQ.

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_1
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_2
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_3
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_4
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_5
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_6
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_7
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_8
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_9
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_10
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_11
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_12
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_13
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_14
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_15
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_16
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_17
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_18
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_19
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_20
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_21
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_22
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_23
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_24
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_25
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_26
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_27
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_28
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_29
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_30
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_31
  - LL\_RCC\_PLLSAIDIVQ\_DIV\_32

### Notes

- used PLLSAICLK selected (SAI clock)

### Reference Manual to LL API cross reference:

- DCKCFGR PLLSAIDIVQ LL\_RCC\_PLLSAI\_GetDIVQ

### **LL\_RCC\_PLLSAI\_GetDIVR**

### Function name

**\_\_STATIC\_INLINE uint32\_t LL\_RCC\_PLLSAI\_GetDIVR (void )**

### Function description

Get SAIPLL division factor for PLLSAIDIVR.



#### Return values

- **Returned:** value can be one of the following values:
  - LL\_RCC\_PLLSAIDIVR\_DIV\_2
  - LL\_RCC\_PLLSAIDIVR\_DIV\_4
  - LL\_RCC\_PLLSAIDIVR\_DIV\_8
  - LL\_RCC\_PLLSAIDIVR\_DIV\_16

#### Notes

- used for LTDC domain clock

#### Reference Manual to LL API cross reference:

- DCKCFGR PLLSAIDIVR LL\_RCC\_PLLSAI\_GetDIVR

#### 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:

- CIR 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:

- CIR 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:

- CIR 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:

- CIR 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:

- CIR PLLRDYC LL\_RCC\_ClearFlag\_PLLRDY

### LL\_RCC\_ClearFlag\_PLLI2SRDY

#### Function name

```
__STATIC_INLINE void LL_RCC_ClearFlag_PLLI2SRDY (void )
```

#### Function description

Clear PLLI2S ready interrupt flag.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CIR PLLI2SRDYC LL\_RCC\_ClearFlag\_PLLI2SRDY

### LL\_RCC\_ClearFlag\_PLLSAIRDY

#### Function name

```
__STATIC_INLINE void LL_RCC_ClearFlag_PLLSAIRDY (void )
```

#### Function description

Clear PLLSAI ready interrupt flag.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CIR PLLSAIRDYC LL\_RCC\_ClearFlag\_PLLSAIRDY

### 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:**

- CIR CSSC LL\_RCC\_ClearFlag\_HSECSS

### 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:**

- CIR 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:**

- CIR 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:**

- CIR 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:

- CIR 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:

- CIR PLLRDYF LL\_RCC\_IsActiveFlag\_PLLRDY

### LL\_RCC\_IsActiveFlag\_PLLI2SRDY

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLI2SRDY (void )`

#### Function description

Check if PLLI2S ready interrupt occurred or not.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CIR PLLI2SRDYF LL\_RCC\_IsActiveFlag\_PLLI2SRDY

### LL\_RCC\_IsActiveFlag\_PLLSAIRDY

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLLSAIRDY (void )`

#### Function description

Check if PLLSAI ready interrupt occurred or not.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CIR PLLSAIRDYF LL\_RCC\_IsActiveFlag\_PLLSAIRDY

### 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:

- CIR CSSF LL\_RCC\_IsActiveFlag\_HSECSS

### 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\_LPWRST

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRST (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\_LPWRST

### 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\_PORRST

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST (void )`

#### Function description

Check if RCC flag POR/PDR reset is set or not.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CSR PORRSTF LL\_RCC\_IsActiveFlag\_PORRST

### 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:**

- CIR 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:**

- CIR 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:**

- CIR 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:**

- CIR 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:**

- CIR PLLRDYIE LL\_RCC\_EnableIT\_PLLRDY

### LL\_RCC\_EnableIT\_PLLI2SRDY

**Function name**

`__STATIC_INLINE void LL_RCC_EnableIT_PLLI2SRDY (void )`

**Function description**

Enable PLLI2S ready interrupt.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CIR PLLI2SRDYIE LL\_RCC\_EnableIT\_PLLI2SRDY

### LL\_RCC\_EnableIT\_PLLSAIRDY

**Function name**

`__STATIC_INLINE void LL_RCC_EnableIT_PLLSAIRDY (void )`

**Function description**

Enable PLLSAI ready interrupt.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CIR PLLSAIRDYIE LL\_RCC\_EnableIT\_PLLSAIRDY



### 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:**

- CIR 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:**

- CIR LSE RDY IE 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:**

- CIR 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:**

- CIR 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:

- CIR PLLRDYIE LL\_RCC\_DisableIT\_PLLRDY

### LL\_RCC\_DisableIT\_PLLI2SRDY

#### Function name

`__STATIC_INLINE void LL_RCC_DisableIT_PLLI2SRDY (void )`

#### Function description

Disable PLLI2S ready interrupt.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CIR PLLI2SRDYIE LL\_RCC\_DisableIT\_PLLI2SRDY

### LL\_RCC\_DisableIT\_PLLSAIRDY

#### Function name

`__STATIC_INLINE void LL_RCC_DisableIT_PLLSAIRDY (void )`

#### Function description

Disable PLLSAI ready interrupt.

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CIR PLLSAIRDYIE LL\_RCC\_DisableIT\_PLLSAIRDY

### 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:

- CIR 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:

- CIR 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:

- CIR 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:

- CIR 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:

- CIR PLLRDYIE LL\_RCC\_IsEnabledIT\_PLLRDY

### LL\_RCC\_IsEnabledIT\_PLLI2SRDY

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLI2SRDY (void )`

#### Function description

Checks if PLLI2S ready interrupt source is enabled or disabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CIR PLLI2SRDYIE LL\_RCC\_IsEnabledIT\_PLLI2SRDY

### LL\_RCC\_IsEnabledIT\_PLLSAIRDY

#### Function name

`__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLLSAIRDY (void )`

#### Function description

Checks if PLLSAI ready interrupt source is enabled or disabled.

#### Return values

- **State:** of bit (1 or 0).

#### Reference Manual to LL API cross reference:

- CIR PLLSAIRDYIE LL\_RCC\_IsEnabledIT\_PLLSAIRDY

### 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 source HSE and PLL OFF AHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFF All interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocks LSI, 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\_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 (\*)
  - LL\_RCC\_SAI2\_CLKSOURCE (\*)
  - LL\_RCC\_SAI1\_A\_CLKSOURCE (\*)
  - LL\_RCC\_SAI1\_B\_CLKSOURCE (\*)
 (\*) value not defined in all devices.

### Return values

- **SAI:** clock frequency (in Hz)
  - LL\_RCC\_PERIPH\_FREQUENCY\_NO indicates that oscillator is not ready

### LL\_RCC\_GetSDIOClockFreq

#### Function name

**uint32\_t LL\_RCC\_GetSDIOClockFreq (uint32\_t SDIOxSource)**

#### Function description

Return SDIOx clock frequency.

#### Parameters

- **SDIOxSource:** This parameter can be one of the following values:
  - LL\_RCC\_SDIO\_CLKSOURCE

### Return values

- **SDIO:** clock frequency (in Hz)
  - LL\_RCC\_PERIPH\_FREQUENCY\_NO indicates that oscillator is not ready

### 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 is not ready

#### 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 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\_I2S1\_CLKSOURCE
  - LL\_RCC\_I2S2\_CLKSOURCE (\*)
 (\*) value not defined in all devices.

#### Return values

- **I2S:** clock frequency (in Hz)
  - LL\_RCC\_PERIPH\_FREQUENCY\_NO indicates that oscillator is not ready

#### LL\_RCC\_GetLTDClockFreq

#### Function name

**uint32\_t LL\_RCC\_GetLTDClockFreq (uint32\_t LTDCxSource)**

#### Function description

Return LTDC clock frequency.

#### Parameters

- **LTDCxSource:** This parameter can be one of the following values:
  - LL\_RCC\_LTDC\_CLKSOURCE

#### Return values

- **LTDC:** clock frequency (in Hz)
  - LL\_RCC\_PERIPH\_FREQUENCY\_NO indicates that oscillator PLLSAI is not ready

## LL\_RCC\_GetDSIClockFreq

### Function name

**uint32\_t LL\_RCC\_GetDSIClockFreq (uint32\_t DSISource)**

### Function description

Return DSI clock frequency.

### Parameters

- **DSISource:** This parameter can be one of the following values:
  - LL\_RCC\_DSI\_CLKSOURCE

### Return values

- **DSI:** clock frequency (in Hz)
  - LL\_RCC\_PERIPH\_FREQUENCY\_NO indicates that oscillator is not ready
  - LL\_RCC\_PERIPH\_FREQUENCY\_NA indicates that external clock is used

## 87.3 RCC Firmware driver defines

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

### 87.3.1 RCC

RCC

***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

***Peripheral CK48M get clock source***

#### LL\_RCC\_CK48M\_CLKSOURCE

CK48M Domain clock source selection  
**Peripheral 48Mhz domain clock source selection**

#### LL\_RCC\_CK48M\_CLKSOURCE\_PLL

PLL oscillator clock used as 48Mhz domain clock

#### LL\_RCC\_CK48M\_CLKSOURCE\_PLLSAI

PLLSAI oscillator clock used as 48Mhz domain clock  
**Clear Flags Defines**

#### LL\_RCC\_CIR\_LSIRDYC

LSI Ready Interrupt Clear

#### LL\_RCC\_CIR\_LSERDYC

LSE Ready Interrupt Clear

#### LL\_RCC\_CIR\_HSIRDYC

HSI Ready Interrupt Clear

#### LL\_RCC\_CIR\_HSERDYC

HSE Ready Interrupt Clear

#### LL\_RCC\_CIR\_PLLRDYC

PLL Ready Interrupt Clear

#### LL\_RCC\_CIR\_PLLI2SRDYC

PLLI2S Ready Interrupt Clear

#### LL\_RCC\_CIR\_PLLSAIRDYC

PLLSAI Ready Interrupt Clear

#### LL\_RCC\_CIR\_CSSC

Clock Security System Interrupt Clear  
**Peripheral DSI get clock source**

#### LL\_RCC\_DSI\_CLKSOURCE

DSI Clock source selection  
**Peripheral DSI clock source selection**

#### LL\_RCC\_DSI\_CLKSOURCE\_PHY

DSI-PHY clock used as DSI byte lane clock source

#### LL\_RCC\_DSI\_CLKSOURCE\_PLL

PLL clock used as DSI byte lane clock source  
**Get Flags Defines**

#### LL\_RCC\_CIR\_LSIRDYF

LSI Ready Interrupt flag

#### LL\_RCC\_CIR\_LSERDYF

LSE Ready Interrupt flag

#### LL\_RCC\_CIR\_HSIRDYF

HSI Ready Interrupt flag



**LL\_RCC\_CIR\_HSERDYF**

HSE Ready Interrupt flag

**LL\_RCC\_CIR\_PLLRDYF**

PLL Ready Interrupt flag

**LL\_RCC\_CIR\_PLLI2SRDYF**

PLLI2S Ready Interrupt flag

**LL\_RCC\_CIR\_PLLSAIRDYF**

PLLSAI Ready Interrupt flag

**LL\_RCC\_CIR\_CSSF**

Clock Security System Interrupt flag

**LL\_RCC\_CSR\_LPWRSTF**

Low-Power reset flag

**LL\_RCC\_CSR\_PINRSTF**

PIN reset flag

**LL\_RCC\_CSR\_PORRSTF**

POR/PDR 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 I2S get clock source***

**LL\_RCC\_I2S1\_CLKSOURCE**

I2S1 Clock source selection

***Peripheral I2S clock source selection***

**LL\_RCC\_I2S1\_CLKSOURCE\_PLLI2S**

I2S oscillator clock used as I2S1 clock

**LL\_RCC\_I2S1\_CLKSOURCE\_PIN**

External pin clock used as I2S1 clock

***IT Defines***

**LL\_RCC\_CIR\_LSIRDYIE**

LSI Ready Interrupt Enable

**LL\_RCC\_CIR\_LSERDYIE**

LSE Ready Interrupt Enable

**LL\_RCC\_CIR\_HSIRDYIE**

HSI Ready Interrupt Enable

**LL\_RCC\_CIR\_HSERDYIE**

HSE Ready Interrupt Enable

**LL\_RCC\_CIR\_PLLRDYIE**

PLL Ready Interrupt Enable

**LL\_RCC\_CIR\_PLLI2SRDYIE**

PLLI2S Ready Interrupt Enable

**LL\_RCC\_CIR\_PLLSAIRDYIE**

PLLSAI Ready Interrupt Enable

**Peripheral LTDC get clock source**

**LL\_RCC\_LTDC\_CLKSOURCE**

LTDC Clock source selection

**MCO source selection**

**LL\_RCC\_MCO1SOURCE\_HSI**

HSI selection as MCO1 source

**LL\_RCC\_MCO1SOURCE\_LSE**

LSE selection as MCO1 source

**LL\_RCC\_MCO1SOURCE\_HSE**

HSE selection as MCO1 source

**LL\_RCC\_MCO1SOURCE\_PLLCLK**

PLLCLK selection as MCO1 source

**LL\_RCC\_MCO2SOURCE\_SYSCLK**

SYSCLK selection as MCO2 source

**LL\_RCC\_MCO2SOURCE\_PLLI2S**

PLLI2S selection as MCO2 source

**LL\_RCC\_MCO2SOURCE\_HSE**

HSE selection as MCO2 source

**LL\_RCC\_MCO2SOURCE\_PLLCLK**

PLLCLK selection as MCO2 source

**MCO prescaler**

**LL\_RCC\_MCO1\_DIV\_1**

MCO1 not divided

**LL\_RCC\_MCO1\_DIV\_2**

MCO1 divided by 2

**LL\_RCC\_MCO1\_DIV\_3**

MCO1 divided by 3

**LL\_RCC\_MCO1\_DIV\_4**

MCO1 divided by 4

**LL\_RCC\_MCO1\_DIV\_5**

MCO1 divided by 5

**LL\_RCC\_MCO2\_DIV\_1**

MCO2 not divided

**LL\_RCC\_MCO2\_DIV\_2**

MCO2 divided by 2

**LL\_RCC\_MCO2\_DIV\_3**

MCO2 divided by 3

**LL\_RCC\_MCO2\_DIV\_4**

MCO2 divided by 4

**LL\_RCC\_MCO2\_DIV\_5**

MCO2 divided by 5

***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

**EXTERNAL\_CLOCK\_VALUE**

Value of the I2S\_CKIN external oscillator 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

***PLLI2SDIVQ division factor (PLLI2SDIVQ)***

**LL\_RCC\_PLLI2SDIVQ\_DIV\_1**

PLLI2S division factor for PLLI2SDIVQ output by 1

**LL\_RCC\_PLLI2SDIVQ\_DIV\_2**

PLLI2S division factor for PLLI2SDIVQ output by 2

**LL\_RCC\_PLLI2SDIVQ\_DIV\_3**

PLLI2S division factor for PLLI2SDIVQ output by 3

**LL\_RCC\_PLLI2SDIVQ\_DIV\_4**

PLLI2S division factor for PLLI2SDIVQ output by 4

**LL\_RCC\_PLLI2SDIVQ\_DIV\_5**

PLLI2S division factor for PLLI2SDIVQ output by 5

**LL\_RCC\_PLLI2SDIVQ\_DIV\_6**

PLLI2S division factor for PLLI2SDIVQ output by 6

**LL\_RCC\_PLLI2SDIVQ\_DIV\_7**

PLLI2S division factor for PLLI2SDIVQ output by 7

**LL\_RCC\_PLLI2SDIVQ\_DIV\_8**

PLLI2S division factor for PLLI2SDIVQ output by 8

**LL\_RCC\_PLLI2SDIVQ\_DIV\_9**

PLLI2S division factor for PLLI2SDIVQ output by 9

**LL\_RCC\_PLLI2SDIVQ\_DIV\_10**

PLLI2S division factor for PLLI2SDIVQ output by 10

**LL\_RCC\_PLLI2SDIVQ\_DIV\_11**

PLLI2S division factor for PLLI2SDIVQ output by 11

**LL\_RCC\_PLLI2SDIVQ\_DIV\_12**

PLLI2S division factor for PLLI2SDIVQ output by 12

**LL\_RCC\_PLLI2SDIVQ\_DIV\_13**

PLLI2S division factor for PLLI2SDIVQ output by 13

**LL\_RCC\_PLLI2SDIVQ\_DIV\_14**

PLLI2S division factor for PLLI2SDIVQ output by 14

**LL\_RCC\_PLLI2SDIVQ\_DIV\_15**

PLLI2S division factor for PLLI2SDIVQ output by 15

**LL\_RCC\_PLLI2SDIVQ\_DIV\_16**

PLLI2S division factor for PLLI2SDIVQ output by 16

**LL\_RCC\_PLLI2SDIVQ\_DIV\_17**

PLLI2S division factor for PLLI2SDIVQ output by 17

**LL\_RCC\_PLLI2SDIVQ\_DIV\_18**

PLLI2S division factor for PLLI2SDIVQ output by 18

**LL\_RCC\_PLLI2SDIVQ\_DIV\_19**

PLLI2S division factor for PLLI2SDIVQ output by 19

**LL\_RCC\_PLLI2SDIVQ\_DIV\_20**

PLLI2S division factor for PLLI2SDIVQ output by 20

**LL\_RCC\_PLLI2SDIVQ\_DIV\_21**

PLLI2S division factor for PLLI2SDIVQ output by 21

**LL\_RCC\_PLLI2SDIVQ\_DIV\_22**

PLLI2S division factor for PLLI2SDIVQ output by 22

**LL\_RCC\_PLLI2SDIVQ\_DIV\_23**

PLLI2S division factor for PLLI2SDIVQ output by 23

**LL\_RCC\_PLLI2SDIVQ\_DIV\_24**

PLLI2S division factor for PLLI2SDIVQ output by 24

**LL\_RCC\_PLLI2SDIVQ\_DIV\_25**

PLLI2S division factor for PLLI2SDIVQ output by 25

**LL\_RCC\_PLLI2SDIVQ\_DIV\_26**

PLLI2S division factor for PLLI2SDIVQ output by 26

**LL\_RCC\_PLLI2SDIVQ\_DIV\_27**

PLLI2S division factor for PLLI2SDIVQ output by 27

**LL\_RCC\_PLLI2SDIVQ\_DIV\_28**

PLLI2S division factor for PLLI2SDIVQ output by 28

**LL\_RCC\_PLLI2SDIVQ\_DIV\_29**

PLLI2S division factor for PLLI2SDIVQ output by 29

**LL\_RCC\_PLLI2SDIVQ\_DIV\_30**

PLLI2S division factor for PLLI2SDIVQ output by 30

**LL\_RCC\_PLLI2SDIVQ\_DIV\_31**

PLLI2S division factor for PLLI2SDIVQ output by 31

**LL\_RCC\_PLLI2SDIVQ\_DIV\_32**PLLI2S division factor for PLLI2SDIVQ output by 32  
**PLLI2SM division factor (PLLI2SM)****LL\_RCC\_PLLI2SM\_DIV\_2**

PLLI2S division factor for PLLI2SM output by 2

**LL\_RCC\_PLLI2SM\_DIV\_3**

PLLI2S division factor for PLLI2SM output by 3

**LL\_RCC\_PLLI2SM\_DIV\_4**

PLLI2S division factor for PLLI2SM output by 4

**LL\_RCC\_PLLI2SM\_DIV\_5**

PLLI2S division factor for PLLI2SM output by 5

**LL\_RCC\_PLLI2SM\_DIV\_6**

PLLI2S division factor for PLLI2SM output by 6

**LL\_RCC\_PLLI2SM\_DIV\_7**

PLLI2S division factor for PLLI2SM output by 7

**LL\_RCC\_PLLI2SM\_DIV\_8**

PLLI2S division factor for PLLI2SM output by 8

**LL\_RCC\_PLLI2SM\_DIV\_9**

PLLI2S division factor for PLLI2SM output by 9

**LL\_RCC\_PLLI2SM\_DIV\_10**

PLLI2S division factor for PLLI2SM output by 10

**LL\_RCC\_PLLI2SM\_DIV\_11**

PLLI2S division factor for PLLI2SM output by 11

**LL\_RCC\_PLLI2SM\_DIV\_12**

PLLI2S division factor for PLLI2SM output by 12

**LL\_RCC\_PLLI2SM\_DIV\_13**

PLLI2S division factor for PLLI2SM output by 13

**LL\_RCC\_PLLI2SM\_DIV\_14**

PLLI2S division factor for PLLI2SM output by 14

**LL\_RCC\_PLLI2SM\_DIV\_15**

PLLI2S division factor for PLLI2SM output by 15

**LL\_RCC\_PLLI2SM\_DIV\_16**

PLLI2S division factor for PLLI2SM output by 16

**LL\_RCC\_PLLI2SM\_DIV\_17**

PLLI2S division factor for PLLI2SM output by 17

**LL\_RCC\_PLLI2SM\_DIV\_18**

PLLI2S division factor for PLLI2SM output by 18

**LL\_RCC\_PLLI2SM\_DIV\_19**

PLLI2S division factor for PLLI2SM output by 19

**LL\_RCC\_PLLI2SM\_DIV\_20**

PLLI2S division factor for PLLI2SM output by 20

**LL\_RCC\_PLLI2SM\_DIV\_21**

PLLI2S division factor for PLLI2SM output by 21

**LL\_RCC\_PLLI2SM\_DIV\_22**

PLLI2S division factor for PLLI2SM output by 22

**LL\_RCC\_PLLI2SM\_DIV\_23**

PLLI2S division factor for PLLI2SM output by 23

**LL\_RCC\_PLLI2SM\_DIV\_24**

PLLI2S division factor for PLLI2SM output by 24

**LL\_RCC\_PLLI2SM\_DIV\_25**

PLLI2S division factor for PLLI2SM output by 25

**LL\_RCC\_PLLI2SM\_DIV\_26**

PLLI2S division factor for PLLI2SM output by 26

**LL\_RCC\_PLLI2SM\_DIV\_27**

PLLI2S division factor for PLLI2SM output by 27

**LL\_RCC\_PLLI2SM\_DIV\_28**

PLLI2S division factor for PLLI2SM output by 28

**LL\_RCC\_PLLI2SM\_DIV\_29**

PLLI2S division factor for PLLI2SM output by 29

**LL\_RCC\_PLLI2SM\_DIV\_30**

PLLI2S division factor for PLLI2SM output by 30

**LL\_RCC\_PLLI2SM\_DIV\_31**

PLLI2S division factor for PLLI2SM output by 31

**LL\_RCC\_PLLI2SM\_DIV\_32**

PLLI2S division factor for PLLI2SM output by 32

**LL\_RCC\_PLLI2SM\_DIV\_33**

PLLI2S division factor for PLLI2SM output by 33

**LL\_RCC\_PLLI2SM\_DIV\_34**

PLLI2S division factor for PLLI2SM output by 34

**LL\_RCC\_PLLI2SM\_DIV\_35**

PLLI2S division factor for PLLI2SM output by 35

**LL\_RCC\_PLLI2SM\_DIV\_36**

PLLI2S division factor for PLLI2SM output by 36

**LL\_RCC\_PLLI2SM\_DIV\_37**

PLLI2S division factor for PLLI2SM output by 37

**LL\_RCC\_PLLI2SM\_DIV\_38**

PLLI2S division factor for PLLI2SM output by 38

**LL\_RCC\_PLLI2SM\_DIV\_39**

PLLI2S division factor for PLLI2SM output by 39

**LL\_RCC\_PLLI2SM\_DIV\_40**

PLLI2S division factor for PLLI2SM output by 40

**LL\_RCC\_PLLI2SM\_DIV\_41**

PLLI2S division factor for PLLI2SM output by 41

**LL\_RCC\_PLLI2SM\_DIV\_42**

PLLI2S division factor for PLLI2SM output by 42

**LL\_RCC\_PLLI2SM\_DIV\_43**

PLLI2S division factor for PLLI2SM output by 43

**LL\_RCC\_PLLI2SM\_DIV\_44**

PLLI2S division factor for PLLI2SM output by 44

**LL\_RCC\_PLLI2SM\_DIV\_45**

PLLI2S division factor for PLLI2SM output by 45

**LL\_RCC\_PLLI2SM\_DIV\_46**

PLLI2S division factor for PLLI2SM output by 46

**LL\_RCC\_PLLI2SM\_DIV\_47**

PLLI2S division factor for PLLI2SM output by 47

**LL\_RCC\_PLLI2SM\_DIV\_48**

PLLI2S division factor for PLLI2SM output by 48

**LL\_RCC\_PLLI2SM\_DIV\_49**

PLLI2S division factor for PLLI2SM output by 49

**LL\_RCC\_PLLI2SM\_DIV\_50**

PLLI2S division factor for PLLI2SM output by 50

**LL\_RCC\_PLLI2SM\_DIV\_51**

PLLI2S division factor for PLLI2SM output by 51

**LL\_RCC\_PLLI2SM\_DIV\_52**

PLLI2S division factor for PLLI2SM output by 52

**LL\_RCC\_PLLI2SM\_DIV\_53**

PLLI2S division factor for PLLI2SM output by 53

**LL\_RCC\_PLLI2SM\_DIV\_54**

PLLI2S division factor for PLLI2SM output by 54

**LL\_RCC\_PLLI2SM\_DIV\_55**

PLLI2S division factor for PLLI2SM output by 55

**LL\_RCC\_PLLI2SM\_DIV\_56**

PLLI2S division factor for PLLI2SM output by 56

**LL\_RCC\_PLLI2SM\_DIV\_57**

PLLI2S division factor for PLLI2SM output by 57

**LL\_RCC\_PLLI2SM\_DIV\_58**

PLLI2S division factor for PLLI2SM output by 58

**LL\_RCC\_PLLI2SM\_DIV\_59**

PLLI2S division factor for PLLI2SM output by 59

**LL\_RCC\_PLLI2SM\_DIV\_60**

PLLI2S division factor for PLLI2SM output by 60

**LL\_RCC\_PLLI2SM\_DIV\_61**

PLLI2S division factor for PLLI2SM output by 61

**LL\_RCC\_PLLI2SM\_DIV\_62**

PLLI2S division factor for PLLI2SM output by 62

**LL\_RCC\_PLLI2SM\_DIV\_63**

PLLI2S division factor for PLLI2SM output by 63

***PLLI2SQ division factor (PLLI2SQ)*****LL\_RCC\_PLLI2SQ\_DIV\_2**

PLLI2S division factor for PLLI2SQ output by 2

**LL\_RCC\_PLLI2SQ\_DIV\_3**

PLLI2S division factor for PLLI2SQ output by 3

**LL\_RCC\_PLLI2SQ\_DIV\_4**

PLLI2S division factor for PLLI2SQ output by 4

**LL\_RCC\_PLLI2SQ\_DIV\_5**

PLLI2S division factor for PLLI2SQ output by 5

**LL\_RCC\_PLLI2SQ\_DIV\_6**

PLLI2S division factor for PLLI2SQ output by 6

**LL\_RCC\_PLLI2SQ\_DIV\_7**

PLLI2S division factor for PLLI2SQ output by 7

**LL\_RCC\_PLLI2SQ\_DIV\_8**

PLLI2S division factor for PLLI2SQ output by 8



**LL\_RCC\_PLLI2SQ\_DIV\_9**

PLLI2S division factor for PLLI2SQ output by 9

**LL\_RCC\_PLLI2SQ\_DIV\_10**

PLLI2S division factor for PLLI2SQ output by 10

**LL\_RCC\_PLLI2SQ\_DIV\_11**

PLLI2S division factor for PLLI2SQ output by 11

**LL\_RCC\_PLLI2SQ\_DIV\_12**

PLLI2S division factor for PLLI2SQ output by 12

**LL\_RCC\_PLLI2SQ\_DIV\_13**

PLLI2S division factor for PLLI2SQ output by 13

**LL\_RCC\_PLLI2SQ\_DIV\_14**

PLLI2S division factor for PLLI2SQ output by 14

**LL\_RCC\_PLLI2SQ\_DIV\_15**

PLLI2S division factor for PLLI2SQ output by 15  
**PLLI2SR division factor (PLLI2SR)**

**LL\_RCC\_PLLI2SR\_DIV\_2**

PLLI2S division factor for PLLI2SR output by 2

**LL\_RCC\_PLLI2SR\_DIV\_3**

PLLI2S division factor for PLLI2SR output by 3

**LL\_RCC\_PLLI2SR\_DIV\_4**

PLLI2S division factor for PLLI2SR output by 4

**LL\_RCC\_PLLI2SR\_DIV\_5**

PLLI2S division factor for PLLI2SR output by 5

**LL\_RCC\_PLLI2SR\_DIV\_6**

PLLI2S division factor for PLLI2SR output by 6

**LL\_RCC\_PLLI2SR\_DIV\_7**

PLLI2S division factor for PLLI2SR output by 7  
**PLL, PLLI2S and PLLSAI division factor**

**LL\_RCC\_PLLM\_DIV\_2**

PLL, PLLI2S and PLLSAI division factor by 2

**LL\_RCC\_PLLM\_DIV\_3**

PLL, PLLI2S and PLLSAI division factor by 3

**LL\_RCC\_PLLM\_DIV\_4**

PLL, PLLI2S and PLLSAI division factor by 4

**LL\_RCC\_PLLM\_DIV\_5**

PLL, PLLI2S and PLLSAI division factor by 5

**LL\_RCC\_PLLM\_DIV\_6**

PLL, PLLI2S and PLLSAI division factor by 6

**LL\_RCC\_PLLM\_DIV\_7**

PLL, PLLI2S and PLLSAI division factor by 7

**LL\_RCC\_PLLM\_DIV\_8**

PLL, PLLI2S and PLLSAI division factor by 8

**LL\_RCC\_PLLM\_DIV\_9**

PLL, PLLI2S and PLLSAI division factor by 9

**LL\_RCC\_PLLM\_DIV\_10**

PLL, PLLI2S and PLLSAI division factor by 10

**LL\_RCC\_PLLM\_DIV\_11**

PLL, PLLI2S and PLLSAI division factor by 11

**LL\_RCC\_PLLM\_DIV\_12**

PLL, PLLI2S and PLLSAI division factor by 12

**LL\_RCC\_PLLM\_DIV\_13**

PLL, PLLI2S and PLLSAI division factor by 13

**LL\_RCC\_PLLM\_DIV\_14**

PLL, PLLI2S and PLLSAI division factor by 14

**LL\_RCC\_PLLM\_DIV\_15**

PLL, PLLI2S and PLLSAI division factor by 15

**LL\_RCC\_PLLM\_DIV\_16**

PLL, PLLI2S and PLLSAI division factor by 16

**LL\_RCC\_PLLM\_DIV\_17**

PLL, PLLI2S and PLLSAI division factor by 17

**LL\_RCC\_PLLM\_DIV\_18**

PLL, PLLI2S and PLLSAI division factor by 18

**LL\_RCC\_PLLM\_DIV\_19**

PLL, PLLI2S and PLLSAI division factor by 19

**LL\_RCC\_PLLM\_DIV\_20**

PLL, PLLI2S and PLLSAI division factor by 20

**LL\_RCC\_PLLM\_DIV\_21**

PLL, PLLI2S and PLLSAI division factor by 21

**LL\_RCC\_PLLM\_DIV\_22**

PLL, PLLI2S and PLLSAI division factor by 22

**LL\_RCC\_PLLM\_DIV\_23**

PLL, PLLI2S and PLLSAI division factor by 23

**LL\_RCC\_PLLM\_DIV\_24**

PLL, PLLI2S and PLLSAI division factor by 24

**LL\_RCC\_PLLM\_DIV\_25**

PLL, PLLI2S and PLLSAI division factor by 25

**LL\_RCC\_PLLM\_DIV\_26**

PLL, PLLI2S and PLLSAI division factor by 26

**LL\_RCC\_PLLM\_DIV\_27**

PLL, PLLI2S and PLLSAI division factor by 27

**LL\_RCC\_PLLM\_DIV\_28**

PLL, PLLI2S and PLLSAI division factor by 28

**LL\_RCC\_PLLM\_DIV\_29**

PLL, PLLI2S and PLLSAI division factor by 29

**LL\_RCC\_PLLM\_DIV\_30**

PLL, PLLI2S and PLLSAI division factor by 30

**LL\_RCC\_PLLM\_DIV\_31**

PLL, PLLI2S and PLLSAI division factor by 31

**LL\_RCC\_PLLM\_DIV\_32**

PLL, PLLI2S and PLLSAI division factor by 32

**LL\_RCC\_PLLM\_DIV\_33**

PLL, PLLI2S and PLLSAI division factor by 33

**LL\_RCC\_PLLM\_DIV\_34**

PLL, PLLI2S and PLLSAI division factor by 34

**LL\_RCC\_PLLM\_DIV\_35**

PLL, PLLI2S and PLLSAI division factor by 35

**LL\_RCC\_PLLM\_DIV\_36**

PLL, PLLI2S and PLLSAI division factor by 36

**LL\_RCC\_PLLM\_DIV\_37**

PLL, PLLI2S and PLLSAI division factor by 37

**LL\_RCC\_PLLM\_DIV\_38**

PLL, PLLI2S and PLLSAI division factor by 38

**LL\_RCC\_PLLM\_DIV\_39**

PLL, PLLI2S and PLLSAI division factor by 39

**LL\_RCC\_PLLM\_DIV\_40**

PLL, PLLI2S and PLLSAI division factor by 40

**LL\_RCC\_PLLM\_DIV\_41**

PLL, PLLI2S and PLLSAI division factor by 41

**LL\_RCC\_PLLM\_DIV\_42**

PLL, PLLI2S and PLLSAI division factor by 42

**LL\_RCC\_PLLM\_DIV\_43**

PLL, PLLI2S and PLLSAI division factor by 43

**LL\_RCC\_PLLM\_DIV\_44**

PLL, PLLI2S and PLLSAI division factor by 44

**LL\_RCC\_PLLM\_DIV\_45**

PLL, PLLI2S and PLLSAI division factor by 45

**LL\_RCC\_PLLM\_DIV\_46**

PLL, PLLI2S and PLLSAI division factor by 46

**LL\_RCC\_PLLM\_DIV\_47**

PLL, PLLI2S and PLLSAI division factor by 47

**LL\_RCC\_PLLM\_DIV\_48**

PLL, PLLI2S and PLLSAI division factor by 48

**LL\_RCC\_PLLM\_DIV\_49**

PLL, PLLI2S and PLLSAI division factor by 49

**LL\_RCC\_PLLM\_DIV\_50**

PLL, PLLI2S and PLLSAI division factor by 50

**LL\_RCC\_PLLM\_DIV\_51**

PLL, PLLI2S and PLLSAI division factor by 51

**LL\_RCC\_PLLM\_DIV\_52**

PLL, PLLI2S and PLLSAI division factor by 52

**LL\_RCC\_PLLM\_DIV\_53**

PLL, PLLI2S and PLLSAI division factor by 53

**LL\_RCC\_PLLM\_DIV\_54**

PLL, PLLI2S and PLLSAI division factor by 54

**LL\_RCC\_PLLM\_DIV\_55**

PLL, PLLI2S and PLLSAI division factor by 55

**LL\_RCC\_PLLM\_DIV\_56**

PLL, PLLI2S and PLLSAI division factor by 56

**LL\_RCC\_PLLM\_DIV\_57**

PLL, PLLI2S and PLLSAI division factor by 57

**LL\_RCC\_PLLM\_DIV\_58**

PLL, PLLI2S and PLLSAI division factor by 58

**LL\_RCC\_PLLM\_DIV\_59**

PLL, PLLI2S and PLLSAI division factor by 59

**LL\_RCC\_PLLM\_DIV\_60**

PLL, PLLI2S and PLLSAI division factor by 60

**LL\_RCC\_PLLM\_DIV\_61**

PLL, PLLI2S and PLLSAI division factor by 61

**LL\_RCC\_PLLM\_DIV\_62**

PLL, PLLI2S and PLLSAI division factor by 62

**LL\_RCC\_PLLM\_DIV\_63**

PLL, PLLI2S and PLLSAI division factor by 63

***PLL division factor (PLL<sub>P</sub>)*****LL\_RCC\_PLLP\_DIV\_2**

Main PLL division factor for PLLP output by 2

**LL\_RCC\_PLLP\_DIV\_4**

Main PLL division factor for PLLP output by 4

**LL\_RCC\_PLLP\_DIV\_6**

Main PLL division factor for PLLP output by 6

**LL\_RCC\_PLLP\_DIV\_8**

Main PLL division factor for PLLP output by 8  
**PLL division factor (PLLQ)**

**LL\_RCC\_PLLQ\_DIV\_2**

Main PLL division factor for PLLQ output by 2

**LL\_RCC\_PLLQ\_DIV\_3**

Main PLL division factor for PLLQ output by 3

**LL\_RCC\_PLLQ\_DIV\_4**

Main PLL division factor for PLLQ output by 4

**LL\_RCC\_PLLQ\_DIV\_5**

Main PLL division factor for PLLQ output by 5

**LL\_RCC\_PLLQ\_DIV\_6**

Main PLL division factor for PLLQ output by 6

**LL\_RCC\_PLLQ\_DIV\_7**

Main PLL division factor for PLLQ output by 7

**LL\_RCC\_PLLQ\_DIV\_8**

Main PLL division factor for PLLQ output by 8

**LL\_RCC\_PLLQ\_DIV\_9**

Main PLL division factor for PLLQ output by 9

**LL\_RCC\_PLLQ\_DIV\_10**

Main PLL division factor for PLLQ output by 10

**LL\_RCC\_PLLQ\_DIV\_11**

Main PLL division factor for PLLQ output by 11

**LL\_RCC\_PLLQ\_DIV\_12**

Main PLL division factor for PLLQ output by 12

**LL\_RCC\_PLLQ\_DIV\_13**

Main PLL division factor for PLLQ output by 13

**LL\_RCC\_PLLQ\_DIV\_14**

Main PLL division factor for PLLQ output by 14

**LL\_RCC\_PLLQ\_DIV\_15**

Main PLL division factor for PLLQ output by 15  
**PLL division factor (PLLQ)**

**LL\_RCC\_PLLR\_DIV\_2**

Main PLL division factor for PLLCLK (system clock) by 2

**LL\_RCC\_PLLR\_DIV\_3**

Main PLL division factor for PLLCLK (system clock) by 3

**LL\_RCC\_PLLR\_DIV\_4**

Main PLL division factor for PLLCLK (system clock) by 4

**LL\_RCC\_PLLR\_DIV\_5**

Main PLL division factor for PLLCLK (system clock) by 5

**LL\_RCC\_PLLR\_DIV\_6**

Main PLL division factor for PLLCLK (system clock) by 6

**LL\_RCC\_PLLR\_DIV\_7**

Main PLL division factor for PLLCLK (system clock) by 7

***PLLSAIDIVQ division factor (PLLSAIDIVQ)***

**LL\_RCC\_PLLSAIDIVQ\_DIV\_1**

PLLSAI division factor for PLLSAIDIVQ output by 1

**LL\_RCC\_PLLSAIDIVQ\_DIV\_2**

PLLSAI division factor for PLLSAIDIVQ output by 2

**LL\_RCC\_PLLSAIDIVQ\_DIV\_3**

PLLSAI division factor for PLLSAIDIVQ output by 3

**LL\_RCC\_PLLSAIDIVQ\_DIV\_4**

PLLSAI division factor for PLLSAIDIVQ output by 4

**LL\_RCC\_PLLSAIDIVQ\_DIV\_5**

PLLSAI division factor for PLLSAIDIVQ output by 5

**LL\_RCC\_PLLSAIDIVQ\_DIV\_6**

PLLSAI division factor for PLLSAIDIVQ output by 6

**LL\_RCC\_PLLSAIDIVQ\_DIV\_7**

PLLSAI division factor for PLLSAIDIVQ output by 7

**LL\_RCC\_PLLSAIDIVQ\_DIV\_8**

PLLSAI division factor for PLLSAIDIVQ output by 8

**LL\_RCC\_PLLSAIDIVQ\_DIV\_9**

PLLSAI division factor for PLLSAIDIVQ output by 9

**LL\_RCC\_PLLSAIDIVQ\_DIV\_10**

PLLSAI division factor for PLLSAIDIVQ output by 10

**LL\_RCC\_PLLSAIDIVQ\_DIV\_11**

PLLSAI division factor for PLLSAIDIVQ output by 11

**LL\_RCC\_PLLSAIDIVQ\_DIV\_12**

PLLSAI division factor for PLLSAIDIVQ output by 12

**LL\_RCC\_PLLSAIDIVQ\_DIV\_13**

PLLSAI division factor for PLLSAIDIVQ output by 13

**LL\_RCC\_PLLSAIDIVQ\_DIV\_14**

PLLSAI division factor for PLLSAIDIVQ output by 14

**LL\_RCC\_PLLSAIDIVQ\_DIV\_15**

PLLSAI division factor for PLLSAIDIVQ output by 15

**LL\_RCC\_PLLSAIDIVQ\_DIV\_16**

PLLSAI division factor for PLLSAIDIVQ output by 16

**LL\_RCC\_PLLSAIDIVQ\_DIV\_17**

PLLSAI division factor for PLLSAIDIVQ output by 17

**LL\_RCC\_PLLSAIDIVQ\_DIV\_18**

PLLSAI division factor for PLLSAIDIVQ output by 18

**LL\_RCC\_PLLSAIDIVQ\_DIV\_19**

PLLSAI division factor for PLLSAIDIVQ output by 19

**LL\_RCC\_PLLSAIDIVQ\_DIV\_20**

PLLSAI division factor for PLLSAIDIVQ output by 20

**LL\_RCC\_PLLSAIDIVQ\_DIV\_21**

PLLSAI division factor for PLLSAIDIVQ output by 21

**LL\_RCC\_PLLSAIDIVQ\_DIV\_22**

PLLSAI division factor for PLLSAIDIVQ output by 22

**LL\_RCC\_PLLSAIDIVQ\_DIV\_23**

PLLSAI division factor for PLLSAIDIVQ output by 23

**LL\_RCC\_PLLSAIDIVQ\_DIV\_24**

PLLSAI division factor for PLLSAIDIVQ output by 24

**LL\_RCC\_PLLSAIDIVQ\_DIV\_25**

PLLSAI division factor for PLLSAIDIVQ output by 25

**LL\_RCC\_PLLSAIDIVQ\_DIV\_26**

PLLSAI division factor for PLLSAIDIVQ output by 26

**LL\_RCC\_PLLSAIDIVQ\_DIV\_27**

PLLSAI division factor for PLLSAIDIVQ output by 27

**LL\_RCC\_PLLSAIDIVQ\_DIV\_28**

PLLSAI division factor for PLLSAIDIVQ output by 28

**LL\_RCC\_PLLSAIDIVQ\_DIV\_29**

PLLSAI division factor for PLLSAIDIVQ output by 29

**LL\_RCC\_PLLSAIDIVQ\_DIV\_30**

PLLSAI division factor for PLLSAIDIVQ output by 30

**LL\_RCC\_PLLSAIDIVQ\_DIV\_31**

PLLSAI division factor for PLLSAIDIVQ output by 31

**LL\_RCC\_PLLSAIDIVQ\_DIV\_32**

PLLSAI division factor for PLLSAIDIVQ output by 32

***PLLSAIDIVR division factor (PLLSAIDIVR)*****LL\_RCC\_PLLSAIDIVR\_DIV\_2**

PLLSAI division factor for PLLSAIDIVR output by 2

**LL\_RCC\_PLLSAIDIVR\_DIV\_4**

PLLSAI division factor for PLLSAIDIVR output by 4

**LL\_RCC\_PLLSAIDIVR\_DIV\_8**

PLLSAI division factor for PLLSAIDIVR output by 8

**LL\_RCC\_PLLSAIDIVR\_DIV\_16**

PLLSAI division factor for PLLSAIDIVR output by 16

***PLLSAIM division factor (PLLSAIM or PLLM)*****LL\_RCC\_PLLSAIM\_DIV\_2**

PLLSAI division factor for PLLSAIM output by 2

**LL\_RCC\_PLLSAIM\_DIV\_3**

PLLSAI division factor for PLLSAIM output by 3

**LL\_RCC\_PLLSAIM\_DIV\_4**

PLLSAI division factor for PLLSAIM output by 4

**LL\_RCC\_PLLSAIM\_DIV\_5**

PLLSAI division factor for PLLSAIM output by 5

**LL\_RCC\_PLLSAIM\_DIV\_6**

PLLSAI division factor for PLLSAIM output by 6

**LL\_RCC\_PLLSAIM\_DIV\_7**

PLLSAI division factor for PLLSAIM output by 7

**LL\_RCC\_PLLSAIM\_DIV\_8**

PLLSAI division factor for PLLSAIM output by 8

**LL\_RCC\_PLLSAIM\_DIV\_9**

PLLSAI division factor for PLLSAIM output by 9

**LL\_RCC\_PLLSAIM\_DIV\_10**

PLLSAI division factor for PLLSAIM output by 10

**LL\_RCC\_PLLSAIM\_DIV\_11**

PLLSAI division factor for PLLSAIM output by 11

**LL\_RCC\_PLLSAIM\_DIV\_12**

PLLSAI division factor for PLLSAIM output by 12

**LL\_RCC\_PLLSAIM\_DIV\_13**

PLLSAI division factor for PLLSAIM output by 13

**LL\_RCC\_PLLSAIM\_DIV\_14**

PLLSAI division factor for PLLSAIM output by 14

**LL\_RCC\_PLLSAIM\_DIV\_15**

PLLSAI division factor for PLLSAIM output by 15

**LL\_RCC\_PLLSAIM\_DIV\_16**

PLLSAI division factor for PLLSAIM output by 16

**LL\_RCC\_PLLSAIM\_DIV\_17**

PLLSAI division factor for PLLSAIM output by 17

**LL\_RCC\_PLLSAIM\_DIV\_18**

PLLSAI division factor for PLLSAIM output by 18



**LL\_RCC\_PLLSAIM\_DIV\_19**

PLLSAI division factor for PLLSAIM output by 19

**LL\_RCC\_PLLSAIM\_DIV\_20**

PLLSAI division factor for PLLSAIM output by 20

**LL\_RCC\_PLLSAIM\_DIV\_21**

PLLSAI division factor for PLLSAIM output by 21

**LL\_RCC\_PLLSAIM\_DIV\_22**

PLLSAI division factor for PLLSAIM output by 22

**LL\_RCC\_PLLSAIM\_DIV\_23**

PLLSAI division factor for PLLSAIM output by 23

**LL\_RCC\_PLLSAIM\_DIV\_24**

PLLSAI division factor for PLLSAIM output by 24

**LL\_RCC\_PLLSAIM\_DIV\_25**

PLLSAI division factor for PLLSAIM output by 25

**LL\_RCC\_PLLSAIM\_DIV\_26**

PLLSAI division factor for PLLSAIM output by 26

**LL\_RCC\_PLLSAIM\_DIV\_27**

PLLSAI division factor for PLLSAIM output by 27

**LL\_RCC\_PLLSAIM\_DIV\_28**

PLLSAI division factor for PLLSAIM output by 28

**LL\_RCC\_PLLSAIM\_DIV\_29**

PLLSAI division factor for PLLSAIM output by 29

**LL\_RCC\_PLLSAIM\_DIV\_30**

PLLSAI division factor for PLLSAIM output by 30

**LL\_RCC\_PLLSAIM\_DIV\_31**

PLLSAI division factor for PLLSAIM output by 31

**LL\_RCC\_PLLSAIM\_DIV\_32**

PLLSAI division factor for PLLSAIM output by 32

**LL\_RCC\_PLLSAIM\_DIV\_33**

PLLSAI division factor for PLLSAIM output by 33

**LL\_RCC\_PLLSAIM\_DIV\_34**

PLLSAI division factor for PLLSAIM output by 34

**LL\_RCC\_PLLSAIM\_DIV\_35**

PLLSAI division factor for PLLSAIM output by 35

**LL\_RCC\_PLLSAIM\_DIV\_36**

PLLSAI division factor for PLLSAIM output by 36

**LL\_RCC\_PLLSAIM\_DIV\_37**

PLLSAI division factor for PLLSAIM output by 37

**LL\_RCC\_PLLSAIM\_DIV\_38**

PLLSAI division factor for PLLSAIM output by 38

**LL\_RCC\_PLLSAIM\_DIV\_39**

PLLSAI division factor for PLLSAIM output by 39

**LL\_RCC\_PLLSAIM\_DIV\_40**

PLLSAI division factor for PLLSAIM output by 40

**LL\_RCC\_PLLSAIM\_DIV\_41**

PLLSAI division factor for PLLSAIM output by 41

**LL\_RCC\_PLLSAIM\_DIV\_42**

PLLSAI division factor for PLLSAIM output by 42

**LL\_RCC\_PLLSAIM\_DIV\_43**

PLLSAI division factor for PLLSAIM output by 43

**LL\_RCC\_PLLSAIM\_DIV\_44**

PLLSAI division factor for PLLSAIM output by 44

**LL\_RCC\_PLLSAIM\_DIV\_45**

PLLSAI division factor for PLLSAIM output by 45

**LL\_RCC\_PLLSAIM\_DIV\_46**

PLLSAI division factor for PLLSAIM output by 46

**LL\_RCC\_PLLSAIM\_DIV\_47**

PLLSAI division factor for PLLSAIM output by 47

**LL\_RCC\_PLLSAIM\_DIV\_48**

PLLSAI division factor for PLLSAIM output by 48

**LL\_RCC\_PLLSAIM\_DIV\_49**

PLLSAI division factor for PLLSAIM output by 49

**LL\_RCC\_PLLSAIM\_DIV\_50**

PLLSAI division factor for PLLSAIM output by 50

**LL\_RCC\_PLLSAIM\_DIV\_51**

PLLSAI division factor for PLLSAIM output by 51

**LL\_RCC\_PLLSAIM\_DIV\_52**

PLLSAI division factor for PLLSAIM output by 52

**LL\_RCC\_PLLSAIM\_DIV\_53**

PLLSAI division factor for PLLSAIM output by 53

**LL\_RCC\_PLLSAIM\_DIV\_54**

PLLSAI division factor for PLLSAIM output by 54

**LL\_RCC\_PLLSAIM\_DIV\_55**

PLLSAI division factor for PLLSAIM output by 55

**LL\_RCC\_PLLSAIM\_DIV\_56**

PLLSAI division factor for PLLSAIM output by 56

**LL\_RCC\_PLLSAIM\_DIV\_57**

PLLSAI division factor for PLLSAIM output by 57

**LL\_RCC\_PLLSAIM\_DIV\_58**

PLLSAI division factor for PLLSAIM output by 58

**LL\_RCC\_PLLSAIM\_DIV\_59**

PLLSAI division factor for PLLSAIM output by 59

**LL\_RCC\_PLLSAIM\_DIV\_60**

PLLSAI division factor for PLLSAIM output by 60

**LL\_RCC\_PLLSAIM\_DIV\_61**

PLLSAI division factor for PLLSAIM output by 61

**LL\_RCC\_PLLSAIM\_DIV\_62**

PLLSAI division factor for PLLSAIM output by 62

**LL\_RCC\_PLLSAIM\_DIV\_63**

PLLSAI division factor for PLLSAIM output by 63

***PLLSAIP division factor (PLLSAIP)*****LL\_RCC\_PLLSAIP\_DIV\_2**

PLLSAI division factor for PLLSAIP output by 2

**LL\_RCC\_PLLSAIP\_DIV\_4**

PLLSAI division factor for PLLSAIP output by 4

**LL\_RCC\_PLLSAIP\_DIV\_6**

PLLSAI division factor for PLLSAIP output by 6

**LL\_RCC\_PLLSAIP\_DIV\_8**

PLLSAI division factor for PLLSAIP output by 8

***PLLSAIQ division factor (PLLSAIQ)*****LL\_RCC\_PLLSAIQ\_DIV\_2**

PLLSAI division factor for PLLSAIQ output by 2

**LL\_RCC\_PLLSAIQ\_DIV\_3**

PLLSAI division factor for PLLSAIQ output by 3

**LL\_RCC\_PLLSAIQ\_DIV\_4**

PLLSAI division factor for PLLSAIQ output by 4

**LL\_RCC\_PLLSAIQ\_DIV\_5**

PLLSAI division factor for PLLSAIQ output by 5

**LL\_RCC\_PLLSAIQ\_DIV\_6**

PLLSAI division factor for PLLSAIQ output by 6

**LL\_RCC\_PLLSAIQ\_DIV\_7**

PLLSAI division factor for PLLSAIQ output by 7

**LL\_RCC\_PLLSAIQ\_DIV\_8**

PLLSAI division factor for PLLSAIQ output by 8

**LL\_RCC\_PLLSAIQ\_DIV\_9**

PLLSAI division factor for PLLSAIQ output by 9

**LL\_RCC\_PLLSAIQ\_DIV\_10**

PLLSAI division factor for PLLSAIQ output by 10

**LL\_RCC\_PLLSAIQ\_DIV\_11**

PLLSAI division factor for PLLSAIQ output by 11

**LL\_RCC\_PLLSAIQ\_DIV\_12**

PLLSAI division factor for PLLSAIQ output by 12

**LL\_RCC\_PLLSAIQ\_DIV\_13**

PLLSAI division factor for PLLSAIQ output by 13

**LL\_RCC\_PLLSAIQ\_DIV\_14**

PLLSAI division factor for PLLSAIQ output by 14

**LL\_RCC\_PLLSAIQ\_DIV\_15**

PLLSAI division factor for PLLSAIQ output by 15  
**PLLSAIR division factor (PLLSAIR)**

**LL\_RCC\_PLLSAIR\_DIV\_2**

PLLSAI division factor for PLLSAIR output by 2

**LL\_RCC\_PLLSAIR\_DIV\_3**

PLLSAI division factor for PLLSAIR output by 3

**LL\_RCC\_PLLSAIR\_DIV\_4**

PLLSAI division factor for PLLSAIR output by 4

**LL\_RCC\_PLLSAIR\_DIV\_5**

PLLSAI division factor for PLLSAIR output by 5

**LL\_RCC\_PLLSAIR\_DIV\_6**

PLLSAI division factor for PLLSAIR output by 6

**LL\_RCC\_PLLSAIR\_DIV\_7**

PLLSAI division factor for PLLSAIR output by 7  
**PLL, PLLI2S and PLLSAI entry clock source**

**LL\_RCC\_PLLSOURCE\_HSI**

HSI16 clock selected as PLL entry clock source

**LL\_RCC\_PLLSOURCE\_HSE**

HSE clock selected as PLL entry clock source  
**PLL Spread Spectrum Selection**

**LL\_RCC\_SPREAD\_SELECT\_CENTER**

PLL center spread spectrum selection

**LL\_RCC\_SPREAD\_SELECT\_DOWN**

PLL down spread spectrum selection  
**Peripheral RNG get clock source**

**LL\_RCC\_RNG\_CLKSOURCE**

RNG Clock source selection  
**Peripheral RNG clock source selection**

**LL\_RCC\_RNG\_CLKSOURCE\_PLL**

PLL clock used as RNG clock source

**LL\_RCC\_RNG\_CLKSOURCE\_PLLSAI**

PLLSAI 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**

HSE oscillator clock divided by HSE prescaler used as RTC clock

***HSE prescaler for RTC clock*****LL\_RCC\_RTC\_NOCLOCK**

HSE not divided

**LL\_RCC\_RTC\_HSE\_DIV\_2**

HSE clock divided by 2

**LL\_RCC\_RTC\_HSE\_DIV\_3**

HSE clock divided by 3

**LL\_RCC\_RTC\_HSE\_DIV\_4**

HSE clock divided by 4

**LL\_RCC\_RTC\_HSE\_DIV\_5**

HSE clock divided by 5

**LL\_RCC\_RTC\_HSE\_DIV\_6**

HSE clock divided by 6

**LL\_RCC\_RTC\_HSE\_DIV\_7**

HSE clock divided by 7

**LL\_RCC\_RTC\_HSE\_DIV\_8**

HSE clock divided by 8

**LL\_RCC\_RTC\_HSE\_DIV\_9**

HSE clock divided by 9

**LL\_RCC\_RTC\_HSE\_DIV\_10**

HSE clock divided by 10

**LL\_RCC\_RTC\_HSE\_DIV\_11**

HSE clock divided by 11

**LL\_RCC\_RTC\_HSE\_DIV\_12**

HSE clock divided by 12

**LL\_RCC\_RTC\_HSE\_DIV\_13**

HSE clock divided by 13

**LL\_RCC\_RTC\_HSE\_DIV\_14**

HSE clock divided by 14

**LL\_RCC\_RTC\_HSE\_DIV\_15**

HSE clock divided by 15

**LL\_RCC\_RTC\_HSE\_DIV\_16**

HSE clock divided by 16

**LL\_RCC\_RTC\_HSE\_DIV\_17**

HSE clock divided by 17

**LL\_RCC\_RTC\_HSE\_DIV\_18**

HSE clock divided by 18

**LL\_RCC\_RTC\_HSE\_DIV\_19**

HSE clock divided by 19

**LL\_RCC\_RTC\_HSE\_DIV\_20**

HSE clock divided by 20

**LL\_RCC\_RTC\_HSE\_DIV\_21**

HSE clock divided by 21

**LL\_RCC\_RTC\_HSE\_DIV\_22**

HSE clock divided by 22

**LL\_RCC\_RTC\_HSE\_DIV\_23**

HSE clock divided by 23

**LL\_RCC\_RTC\_HSE\_DIV\_24**

HSE clock divided by 24

**LL\_RCC\_RTC\_HSE\_DIV\_25**

HSE clock divided by 25

**LL\_RCC\_RTC\_HSE\_DIV\_26**

HSE clock divided by 26

**LL\_RCC\_RTC\_HSE\_DIV\_27**

HSE clock divided by 27

**LL\_RCC\_RTC\_HSE\_DIV\_28**

HSE clock divided by 28

**LL\_RCC\_RTC\_HSE\_DIV\_29**

HSE clock divided by 29

**LL\_RCC\_RTC\_HSE\_DIV\_30**

HSE clock divided by 30

**LL\_RCC\_RTC\_HSE\_DIV\_31**

HSE clock divided by 31

***Peripheral SAI get clock source*****LL\_RCC\_SAI1\_A\_CLKSOURCE**

SAI1 block A Clock source selection

**LL\_RCC\_SAI1\_B\_CLKSOURCE**

SAI1 block B Clock source selection  
**Peripheral SAI clock source selection**

**LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLSAI**

PLLSAI clock used as SAI1 block A clock source

**LL\_RCC\_SAI1\_A\_CLKSOURCE\_PLLI2S**

PLLI2S clock used as SAI1 block A clock source

**LL\_RCC\_SAI1\_A\_CLKSOURCE\_PIN**

External pin clock used as SAI1 block A clock source

**LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLSAI**

PLLSAI clock used as SAI1 block B clock source

**LL\_RCC\_SAI1\_B\_CLKSOURCE\_PLLI2S**

PLLI2S clock used as SAI1 block B clock source

**LL\_RCC\_SAI1\_B\_CLKSOURCE\_PIN**

External pin clock used as SAI1 block B clock source  
**Peripheral SDIO get clock source**

**LL\_RCC\_SDIO\_CLKSOURCE**

SDIO Clock source selection  
**Peripheral SDIO clock source selection**

**LL\_RCC\_SDIO\_CLKSOURCE\_PLL48CLK**

PLL 48M domain clock used as SDIO clock

**LL\_RCC\_SDIO\_CLKSOURCE\_SYSCLK**

System clock clock used as SDIO clock  
**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

**Timers clocks prescalers selection****LL\_RCC\_TIM\_PRESCALER\_TWICE**

Timers clock to twice PCLK

**LL\_RCC\_TIM\_PRESCALER\_FOUR\_TIMES**

Timers clock to four time PCLK

**Peripheral USB get clock source****LL\_RCC\_USB\_CLKSOURCE**

USB Clock source selection

**Peripheral USB clock source selection****LL\_RCC\_USB\_CLKSOURCE\_PLL**

PLL clock used as USB clock source

**LL\_RCC\_USB\_CLKSOURCE\_PLLSAI**

PLLSAI 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_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`
  - `LL_RCC_PLLM_DIV_17`
  - `LL_RCC_PLLM_DIV_18`
  - `LL_RCC_PLLM_DIV_19`
  - `LL_RCC_PLLM_DIV_20`
  - `LL_RCC_PLLM_DIV_21`
  - `LL_RCC_PLLM_DIV_22`
  - `LL_RCC_PLLM_DIV_23`
  - `LL_RCC_PLLM_DIV_24`
  - `LL_RCC_PLLM_DIV_25`
  - `LL_RCC_PLLM_DIV_26`
  - `LL_RCC_PLLM_DIV_27`
  - `LL_RCC_PLLM_DIV_28`
  - `LL_RCC_PLLM_DIV_29`
  - `LL_RCC_PLLM_DIV_30`
  - `LL_RCC_PLLM_DIV_31`
  - `LL_RCC_PLLM_DIV_32`
  - `LL_RCC_PLLM_DIV_33`
  - `LL_RCC_PLLM_DIV_34`
  - `LL_RCC_PLLM_DIV_35`
  - `LL_RCC_PLLM_DIV_36`
  - `LL_RCC_PLLM_DIV_37`
  - `LL_RCC_PLLM_DIV_38`
  - `LL_RCC_PLLM_DIV_39`
  - `LL_RCC_PLLM_DIV_40`
  - `LL_RCC_PLLM_DIV_41`
  - `LL_RCC_PLLM_DIV_42`
  - `LL_RCC_PLLM_DIV_43`
  - `LL_RCC_PLLM_DIV_44`
  - `LL_RCC_PLLM_DIV_45`
  - `LL_RCC_PLLM_DIV_46`
  - `LL_RCC_PLLM_DIV_47`
  - `LL_RCC_PLLM_DIV_48`
  - `LL_RCC_PLLM_DIV_49`
  - `LL_RCC_PLLM_DIV_50`
  - `LL_RCC_PLLM_DIV_51`

**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_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_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`
  - `LL_RCC_PLLM_DIV_17`
  - `LL_RCC_PLLM_DIV_18`
  - `LL_RCC_PLLM_DIV_19`
  - `LL_RCC_PLLM_DIV_20`
  - `LL_RCC_PLLM_DIV_21`
  - `LL_RCC_PLLM_DIV_22`
  - `LL_RCC_PLLM_DIV_23`
  - `LL_RCC_PLLM_DIV_24`
  - `LL_RCC_PLLM_DIV_25`
  - `LL_RCC_PLLM_DIV_26`
  - `LL_RCC_PLLM_DIV_27`
  - `LL_RCC_PLLM_DIV_28`
  - `LL_RCC_PLLM_DIV_29`
  - `LL_RCC_PLLM_DIV_30`
  - `LL_RCC_PLLM_DIV_31`
  - `LL_RCC_PLLM_DIV_32`
  - `LL_RCC_PLLM_DIV_33`
  - `LL_RCC_PLLM_DIV_34`
  - `LL_RCC_PLLM_DIV_35`
  - `LL_RCC_PLLM_DIV_36`
  - `LL_RCC_PLLM_DIV_37`
  - `LL_RCC_PLLM_DIV_38`
  - `LL_RCC_PLLM_DIV_39`
  - `LL_RCC_PLLM_DIV_40`
  - `LL_RCC_PLLM_DIV_41`
  - `LL_RCC_PLLM_DIV_42`
  - `LL_RCC_PLLM_DIV_43`
  - `LL_RCC_PLLM_DIV_44`
  - `LL_RCC_PLLM_DIV_45`
  - `LL_RCC_PLLM_DIV_46`
  - `LL_RCC_PLLM_DIV_47`
  - `LL_RCC_PLLM_DIV_48`
  - `LL_RCC_PLLM_DIV_49`
  - `LL_RCC_PLLM_DIV_50`
  - `LL_RCC_PLLM_DIV_51`

**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_PLLCLK_DSI_FREQ`

**Description:**

- Helper macro to calculate the PLLCLK frequency used on DSI.

**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLL__`: This parameter can be one of the following values:
  - `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`
  - `LL_RCC_PLLM_DIV_17`
  - `LL_RCC_PLLM_DIV_18`
  - `LL_RCC_PLLM_DIV_19`
  - `LL_RCC_PLLM_DIV_20`
  - `LL_RCC_PLLM_DIV_21`
  - `LL_RCC_PLLM_DIV_22`
  - `LL_RCC_PLLM_DIV_23`
  - `LL_RCC_PLLM_DIV_24`
  - `LL_RCC_PLLM_DIV_25`
  - `LL_RCC_PLLM_DIV_26`
  - `LL_RCC_PLLM_DIV_27`
  - `LL_RCC_PLLM_DIV_28`
  - `LL_RCC_PLLM_DIV_29`
  - `LL_RCC_PLLM_DIV_30`
  - `LL_RCC_PLLM_DIV_31`
  - `LL_RCC_PLLM_DIV_32`
  - `LL_RCC_PLLM_DIV_33`
  - `LL_RCC_PLLM_DIV_34`
  - `LL_RCC_PLLM_DIV_35`
  - `LL_RCC_PLLM_DIV_36`
  - `LL_RCC_PLLM_DIV_37`
  - `LL_RCC_PLLM_DIV_38`
  - `LL_RCC_PLLM_DIV_39`
  - `LL_RCC_PLLM_DIV_40`
  - `LL_RCC_PLLM_DIV_41`
  - `LL_RCC_PLLM_DIV_42`
  - `LL_RCC_PLLM_DIV_43`
  - `LL_RCC_PLLM_DIV_44`
  - `LL_RCC_PLLM_DIV_45`
  - `LL_RCC_PLLM_DIV_46`
  - `LL_RCC_PLLM_DIV_47`
  - `LL_RCC_PLLM_DIV_48`
  - `LL_RCC_PLLM_DIV_49`
  - `LL_RCC_PLLM_DIV_50`
  - `LL_RCC_PLLM_DIV_51`



**Return value:**

- PLL: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLCLK_DSI_FREQ (HSE_VALUE, LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetR ());`

## `__LL_RCC_CALC_PLLCLK_SAI_FREQ`

**Description:**

- Helper macro to calculate the PLLCLK frequency used on SAI.

**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLLM__`: This parameter can be one of the following values:
  - `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`
  - `LL_RCC_PLLM_DIV_17`
  - `LL_RCC_PLLM_DIV_18`
  - `LL_RCC_PLLM_DIV_19`
  - `LL_RCC_PLLM_DIV_20`
  - `LL_RCC_PLLM_DIV_21`
  - `LL_RCC_PLLM_DIV_22`
  - `LL_RCC_PLLM_DIV_23`
  - `LL_RCC_PLLM_DIV_24`
  - `LL_RCC_PLLM_DIV_25`
  - `LL_RCC_PLLM_DIV_26`
  - `LL_RCC_PLLM_DIV_27`
  - `LL_RCC_PLLM_DIV_28`
  - `LL_RCC_PLLM_DIV_29`
  - `LL_RCC_PLLM_DIV_30`
  - `LL_RCC_PLLM_DIV_31`
  - `LL_RCC_PLLM_DIV_32`
  - `LL_RCC_PLLM_DIV_33`
  - `LL_RCC_PLLM_DIV_34`
  - `LL_RCC_PLLM_DIV_35`
  - `LL_RCC_PLLM_DIV_36`
  - `LL_RCC_PLLM_DIV_37`
  - `LL_RCC_PLLM_DIV_38`
  - `LL_RCC_PLLM_DIV_39`
  - `LL_RCC_PLLM_DIV_40`
  - `LL_RCC_PLLM_DIV_41`
  - `LL_RCC_PLLM_DIV_42`
  - `LL_RCC_PLLM_DIV_43`
  - `LL_RCC_PLLM_DIV_44`
  - `LL_RCC_PLLM_DIV_45`
  - `LL_RCC_PLLM_DIV_46`
  - `LL_RCC_PLLM_DIV_47`
  - `LL_RCC_PLLM_DIV_48`
  - `LL_RCC_PLLM_DIV_49`
  - `LL_RCC_PLLM_DIV_50`
  - `LL_RCC_PLLM_DIV_51`

**Return value:**

- PLL: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLCLK_SAI_FREQ (HSE_VALUE, LL_RCC_PLL_GetDivider (), LL_RCC_PLL_GetN (), LL_RCC_PLL_GetR (), LL_RCC_PLL_GetDIVR ());`

## `__LL_RCC_CALC_PLLSAI_SAI_FREQ`

**Description:**

- Helper macro to calculate the PLLSAI frequency used for SAI domain.

**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLLM__`: This parameter can be one of the following values:
  - `LL_RCC_PLLSAIM_DIV_2`
  - `LL_RCC_PLLSAIM_DIV_3`
  - `LL_RCC_PLLSAIM_DIV_4`
  - `LL_RCC_PLLSAIM_DIV_5`
  - `LL_RCC_PLLSAIM_DIV_6`
  - `LL_RCC_PLLSAIM_DIV_7`
  - `LL_RCC_PLLSAIM_DIV_8`
  - `LL_RCC_PLLSAIM_DIV_9`
  - `LL_RCC_PLLSAIM_DIV_10`
  - `LL_RCC_PLLSAIM_DIV_11`
  - `LL_RCC_PLLSAIM_DIV_12`
  - `LL_RCC_PLLSAIM_DIV_13`
  - `LL_RCC_PLLSAIM_DIV_14`
  - `LL_RCC_PLLSAIM_DIV_15`
  - `LL_RCC_PLLSAIM_DIV_16`
  - `LL_RCC_PLLSAIM_DIV_17`
  - `LL_RCC_PLLSAIM_DIV_18`
  - `LL_RCC_PLLSAIM_DIV_19`
  - `LL_RCC_PLLSAIM_DIV_20`
  - `LL_RCC_PLLSAIM_DIV_21`
  - `LL_RCC_PLLSAIM_DIV_22`
  - `LL_RCC_PLLSAIM_DIV_23`
  - `LL_RCC_PLLSAIM_DIV_24`
  - `LL_RCC_PLLSAIM_DIV_25`
  - `LL_RCC_PLLSAIM_DIV_26`
  - `LL_RCC_PLLSAIM_DIV_27`
  - `LL_RCC_PLLSAIM_DIV_28`
  - `LL_RCC_PLLSAIM_DIV_29`
  - `LL_RCC_PLLSAIM_DIV_30`
  - `LL_RCC_PLLSAIM_DIV_31`
  - `LL_RCC_PLLSAIM_DIV_32`
  - `LL_RCC_PLLSAIM_DIV_33`
  - `LL_RCC_PLLSAIM_DIV_34`
  - `LL_RCC_PLLSAIM_DIV_35`
  - `LL_RCC_PLLSAIM_DIV_36`
  - `LL_RCC_PLLSAIM_DIV_37`
  - `LL_RCC_PLLSAIM_DIV_38`
  - `LL_RCC_PLLSAIM_DIV_39`
  - `LL_RCC_PLLSAIM_DIV_40`
  - `LL_RCC_PLLSAIM_DIV_41`
  - `LL_RCC_PLLSAIM_DIV_42`
  - `LL_RCC_PLLSAIM_DIV_43`
  - `LL_RCC_PLLSAIM_DIV_44`
  - `LL_RCC_PLLSAIM_DIV_45`
  - `LL_RCC_PLLSAIM_DIV_46`
  - `LL_RCC_PLLSAIM_DIV_47`
  - `LL_RCC_PLLSAIM_DIV_48`
  - `LL_RCC_PLLSAIM_DIV_49`
  - `LL_RCC_PLLSAIM_DIV_50`
  - `LL_RCC_PLLSAIM_DIV_51`

**Return value:**

- PLLSAI: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLSAI_SAI_FREQ (HSE_VALUE,LL_RCC_PLLSAI_GetDivider (), LL_RCC_PLLSAI_GetN (), LL_RCC_PLLSAI_GetQ (), LL_RCC_PLLSAI_GetDIVQ ());`

#### `__LL_RCC_CALC_PLLSAI_48M_FREQ`

**Description:**

- Helper macro to calculate the PLLSAI frequency used on 48Mhz domain.



**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLLM__`: This parameter can be one of the following values:
  - `LL_RCC_PLLSAIM_DIV_2`
  - `LL_RCC_PLLSAIM_DIV_3`
  - `LL_RCC_PLLSAIM_DIV_4`
  - `LL_RCC_PLLSAIM_DIV_5`
  - `LL_RCC_PLLSAIM_DIV_6`
  - `LL_RCC_PLLSAIM_DIV_7`
  - `LL_RCC_PLLSAIM_DIV_8`
  - `LL_RCC_PLLSAIM_DIV_9`
  - `LL_RCC_PLLSAIM_DIV_10`
  - `LL_RCC_PLLSAIM_DIV_11`
  - `LL_RCC_PLLSAIM_DIV_12`
  - `LL_RCC_PLLSAIM_DIV_13`
  - `LL_RCC_PLLSAIM_DIV_14`
  - `LL_RCC_PLLSAIM_DIV_15`
  - `LL_RCC_PLLSAIM_DIV_16`
  - `LL_RCC_PLLSAIM_DIV_17`
  - `LL_RCC_PLLSAIM_DIV_18`
  - `LL_RCC_PLLSAIM_DIV_19`
  - `LL_RCC_PLLSAIM_DIV_20`
  - `LL_RCC_PLLSAIM_DIV_21`
  - `LL_RCC_PLLSAIM_DIV_22`
  - `LL_RCC_PLLSAIM_DIV_23`
  - `LL_RCC_PLLSAIM_DIV_24`
  - `LL_RCC_PLLSAIM_DIV_25`
  - `LL_RCC_PLLSAIM_DIV_26`
  - `LL_RCC_PLLSAIM_DIV_27`
  - `LL_RCC_PLLSAIM_DIV_28`
  - `LL_RCC_PLLSAIM_DIV_29`
  - `LL_RCC_PLLSAIM_DIV_30`
  - `LL_RCC_PLLSAIM_DIV_31`
  - `LL_RCC_PLLSAIM_DIV_32`
  - `LL_RCC_PLLSAIM_DIV_33`
  - `LL_RCC_PLLSAIM_DIV_34`
  - `LL_RCC_PLLSAIM_DIV_35`
  - `LL_RCC_PLLSAIM_DIV_36`
  - `LL_RCC_PLLSAIM_DIV_37`
  - `LL_RCC_PLLSAIM_DIV_38`
  - `LL_RCC_PLLSAIM_DIV_39`
  - `LL_RCC_PLLSAIM_DIV_40`
  - `LL_RCC_PLLSAIM_DIV_41`
  - `LL_RCC_PLLSAIM_DIV_42`
  - `LL_RCC_PLLSAIM_DIV_43`
  - `LL_RCC_PLLSAIM_DIV_44`
  - `LL_RCC_PLLSAIM_DIV_45`
  - `LL_RCC_PLLSAIM_DIV_46`
  - `LL_RCC_PLLSAIM_DIV_47`
  - `LL_RCC_PLLSAIM_DIV_48`
  - `LL_RCC_PLLSAIM_DIV_49`
  - `LL_RCC_PLLSAIM_DIV_50`
  - `LL_RCC_PLLSAIM_DIV_51`

**Return value:**

- PLLSAI: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLSAI_48M_FREQ (HSE_VALUE,LL_RCC_PLLSAI_GetDivider (), LL_RCC_PLLSAI_GetN (), LL_RCC_PLLSAI_GetP ());`

## `__LL_RCC_CALC_PLLSAI_LTDC_FREQ`

**Description:**

- Helper macro to calculate the PLLSAI frequency used for LTDC domain.

**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLLM__`: This parameter can be one of the following values:
  - `LL_RCC_PLLSAIM_DIV_2`
  - `LL_RCC_PLLSAIM_DIV_3`
  - `LL_RCC_PLLSAIM_DIV_4`
  - `LL_RCC_PLLSAIM_DIV_5`
  - `LL_RCC_PLLSAIM_DIV_6`
  - `LL_RCC_PLLSAIM_DIV_7`
  - `LL_RCC_PLLSAIM_DIV_8`
  - `LL_RCC_PLLSAIM_DIV_9`
  - `LL_RCC_PLLSAIM_DIV_10`
  - `LL_RCC_PLLSAIM_DIV_11`
  - `LL_RCC_PLLSAIM_DIV_12`
  - `LL_RCC_PLLSAIM_DIV_13`
  - `LL_RCC_PLLSAIM_DIV_14`
  - `LL_RCC_PLLSAIM_DIV_15`
  - `LL_RCC_PLLSAIM_DIV_16`
  - `LL_RCC_PLLSAIM_DIV_17`
  - `LL_RCC_PLLSAIM_DIV_18`
  - `LL_RCC_PLLSAIM_DIV_19`
  - `LL_RCC_PLLSAIM_DIV_20`
  - `LL_RCC_PLLSAIM_DIV_21`
  - `LL_RCC_PLLSAIM_DIV_22`
  - `LL_RCC_PLLSAIM_DIV_23`
  - `LL_RCC_PLLSAIM_DIV_24`
  - `LL_RCC_PLLSAIM_DIV_25`
  - `LL_RCC_PLLSAIM_DIV_26`
  - `LL_RCC_PLLSAIM_DIV_27`
  - `LL_RCC_PLLSAIM_DIV_28`
  - `LL_RCC_PLLSAIM_DIV_29`
  - `LL_RCC_PLLSAIM_DIV_30`
  - `LL_RCC_PLLSAIM_DIV_31`
  - `LL_RCC_PLLSAIM_DIV_32`
  - `LL_RCC_PLLSAIM_DIV_33`
  - `LL_RCC_PLLSAIM_DIV_34`
  - `LL_RCC_PLLSAIM_DIV_35`
  - `LL_RCC_PLLSAIM_DIV_36`
  - `LL_RCC_PLLSAIM_DIV_37`
  - `LL_RCC_PLLSAIM_DIV_38`
  - `LL_RCC_PLLSAIM_DIV_39`
  - `LL_RCC_PLLSAIM_DIV_40`
  - `LL_RCC_PLLSAIM_DIV_41`
  - `LL_RCC_PLLSAIM_DIV_42`
  - `LL_RCC_PLLSAIM_DIV_43`
  - `LL_RCC_PLLSAIM_DIV_44`
  - `LL_RCC_PLLSAIM_DIV_45`
  - `LL_RCC_PLLSAIM_DIV_46`
  - `LL_RCC_PLLSAIM_DIV_47`
  - `LL_RCC_PLLSAIM_DIV_48`
  - `LL_RCC_PLLSAIM_DIV_49`
  - `LL_RCC_PLLSAIM_DIV_50`
  - `LL_RCC_PLLSAIM_DIV_51`

**Return value:**

- PLLSAI: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLSAI_LTDC_FREQ (HSE_VALUE,LL_RCC_PLLSAI_GetDivider (), LL_RCC_PLLSAI_GetN (), LL_RCC_PLLSAI_GetR (), LL_RCC_PLLSAI_GetDIVR ());`

## `__LL_RCC_CALC_PLLI2S_SAI_FREQ`

**Description:**

- Helper macro to calculate the PLLI2S frequency used for SAI domain.

**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLLM__`: This parameter can be one of the following values:
  - `LL_RCC_PLLI2SM_DIV_2`
  - `LL_RCC_PLLI2SM_DIV_3`
  - `LL_RCC_PLLI2SM_DIV_4`
  - `LL_RCC_PLLI2SM_DIV_5`
  - `LL_RCC_PLLI2SM_DIV_6`
  - `LL_RCC_PLLI2SM_DIV_7`
  - `LL_RCC_PLLI2SM_DIV_8`
  - `LL_RCC_PLLI2SM_DIV_9`
  - `LL_RCC_PLLI2SM_DIV_10`
  - `LL_RCC_PLLI2SM_DIV_11`
  - `LL_RCC_PLLI2SM_DIV_12`
  - `LL_RCC_PLLI2SM_DIV_13`
  - `LL_RCC_PLLI2SM_DIV_14`
  - `LL_RCC_PLLI2SM_DIV_15`
  - `LL_RCC_PLLI2SM_DIV_16`
  - `LL_RCC_PLLI2SM_DIV_17`
  - `LL_RCC_PLLI2SM_DIV_18`
  - `LL_RCC_PLLI2SM_DIV_19`
  - `LL_RCC_PLLI2SM_DIV_20`
  - `LL_RCC_PLLI2SM_DIV_21`
  - `LL_RCC_PLLI2SM_DIV_22`
  - `LL_RCC_PLLI2SM_DIV_23`
  - `LL_RCC_PLLI2SM_DIV_24`
  - `LL_RCC_PLLI2SM_DIV_25`
  - `LL_RCC_PLLI2SM_DIV_26`
  - `LL_RCC_PLLI2SM_DIV_27`
  - `LL_RCC_PLLI2SM_DIV_28`
  - `LL_RCC_PLLI2SM_DIV_29`
  - `LL_RCC_PLLI2SM_DIV_30`
  - `LL_RCC_PLLI2SM_DIV_31`
  - `LL_RCC_PLLI2SM_DIV_32`
  - `LL_RCC_PLLI2SM_DIV_33`
  - `LL_RCC_PLLI2SM_DIV_34`
  - `LL_RCC_PLLI2SM_DIV_35`
  - `LL_RCC_PLLI2SM_DIV_36`
  - `LL_RCC_PLLI2SM_DIV_37`
  - `LL_RCC_PLLI2SM_DIV_38`
  - `LL_RCC_PLLI2SM_DIV_39`
  - `LL_RCC_PLLI2SM_DIV_40`
  - `LL_RCC_PLLI2SM_DIV_41`
  - `LL_RCC_PLLI2SM_DIV_42`
  - `LL_RCC_PLLI2SM_DIV_43`
  - `LL_RCC_PLLI2SM_DIV_44`
  - `LL_RCC_PLLI2SM_DIV_45`
  - `LL_RCC_PLLI2SM_DIV_46`
  - `LL_RCC_PLLI2SM_DIV_47`
  - `LL_RCC_PLLI2SM_DIV_48`
  - `LL_RCC_PLLI2SM_DIV_49`
  - `LL_RCC_PLLI2SM_DIV_50`
  - `LL_RCC_PLLI2SM_DIV_51`

**Return value:**

- PLLI2S: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLI2S_SAI_FREQ (HSE_VALUE,LL_RCC_PLLI2S_GetDivider (),  
LL_RCC_PLLI2S_GetN (), LL_RCC_PLLI2S_GetQ (), LL_RCC_PLLI2S_GetDIVQ ());`



## `__LL_RCC_CALC_PLLI2S_I2S_FREQ`

**Description:**

- Helper macro to calculate the PLLI2S frequency used for I2S domain.

**Parameters:**

- `__INPUTFREQ__`: PLL Input frequency (based on HSE/HSI)
- `__PLLM__`: This parameter can be one of the following values:
  - `LL_RCC_PLLI2SM_DIV_2`
  - `LL_RCC_PLLI2SM_DIV_3`
  - `LL_RCC_PLLI2SM_DIV_4`
  - `LL_RCC_PLLI2SM_DIV_5`
  - `LL_RCC_PLLI2SM_DIV_6`
  - `LL_RCC_PLLI2SM_DIV_7`
  - `LL_RCC_PLLI2SM_DIV_8`
  - `LL_RCC_PLLI2SM_DIV_9`
  - `LL_RCC_PLLI2SM_DIV_10`
  - `LL_RCC_PLLI2SM_DIV_11`
  - `LL_RCC_PLLI2SM_DIV_12`
  - `LL_RCC_PLLI2SM_DIV_13`
  - `LL_RCC_PLLI2SM_DIV_14`
  - `LL_RCC_PLLI2SM_DIV_15`
  - `LL_RCC_PLLI2SM_DIV_16`
  - `LL_RCC_PLLI2SM_DIV_17`
  - `LL_RCC_PLLI2SM_DIV_18`
  - `LL_RCC_PLLI2SM_DIV_19`
  - `LL_RCC_PLLI2SM_DIV_20`
  - `LL_RCC_PLLI2SM_DIV_21`
  - `LL_RCC_PLLI2SM_DIV_22`
  - `LL_RCC_PLLI2SM_DIV_23`
  - `LL_RCC_PLLI2SM_DIV_24`
  - `LL_RCC_PLLI2SM_DIV_25`
  - `LL_RCC_PLLI2SM_DIV_26`
  - `LL_RCC_PLLI2SM_DIV_27`
  - `LL_RCC_PLLI2SM_DIV_28`
  - `LL_RCC_PLLI2SM_DIV_29`
  - `LL_RCC_PLLI2SM_DIV_30`
  - `LL_RCC_PLLI2SM_DIV_31`
  - `LL_RCC_PLLI2SM_DIV_32`
  - `LL_RCC_PLLI2SM_DIV_33`
  - `LL_RCC_PLLI2SM_DIV_34`
  - `LL_RCC_PLLI2SM_DIV_35`
  - `LL_RCC_PLLI2SM_DIV_36`
  - `LL_RCC_PLLI2SM_DIV_37`
  - `LL_RCC_PLLI2SM_DIV_38`
  - `LL_RCC_PLLI2SM_DIV_39`
  - `LL_RCC_PLLI2SM_DIV_40`
  - `LL_RCC_PLLI2SM_DIV_41`
  - `LL_RCC_PLLI2SM_DIV_42`
  - `LL_RCC_PLLI2SM_DIV_43`
  - `LL_RCC_PLLI2SM_DIV_44`
  - `LL_RCC_PLLI2SM_DIV_45`
  - `LL_RCC_PLLI2SM_DIV_46`
  - `LL_RCC_PLLI2SM_DIV_47`
  - `LL_RCC_PLLI2SM_DIV_48`
  - `LL_RCC_PLLI2SM_DIV_49`
  - `LL_RCC_PLLI2SM_DIV_50`
  - `LL_RCC_PLLI2SM_DIV_51`

**Return value:**

- PLLI2S: clock frequency (in Hz)

**Notes:**

- ex: `__LL_RCC_CALC_PLLI2S_I2S_FREQ (HSE_VALUE,LL_RCC_PLLI2S_GetDivider (), LL_RCC_PLLI2S_GetN (), LL_RCC_PLLI2S_GetR ());`

**`__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 (APB1)

**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

## 88 LL RNG Generic Driver

### 88.1 RNG Firmware driver API description

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

#### 88.1.1 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\_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

Return 32-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\_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

## 88.2 RNG Firmware driver defines

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

### 88.2.1 RNG

RNG

#### *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

## 89 LL RTC Generic Driver

### 89.1 RTC Firmware driver registers structures

#### 89.1.1 LL\_RTC\_InitTypeDef

*LL\_RTC\_InitTypeDef* is defined in the `stm32f4xx_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 = 0x7F`This 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 = 0xFFFF`This feature can be modified afterwards using unitary function `LL_RTC_SetSynchPrescaler()`.

#### 89.1.2 LL\_RTC\_TimeTypeDef

*LL\_RTC\_TimeTypeDef* is defined in the `stm32f4xx_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 = 59`This 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 = 59`This feature can be modified afterwards using unitary function `LL_RTC_TIME_SetSecond()`.

#### 89.1.3 LL\_RTC\_DateTypeDef

*LL\_RTC\_DateTypeDef* is defined in the `stm32f4xx_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()`.

**89.1.4**
**LL\_RTC\_AlarmTypeDef**

*LL\_RTC\_AlarmTypeDef* is defined in the `stm32f4xx_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.

**89.2**
**RTC Firmware driver API description**

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

**89.2.1**
**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:

- 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:

- 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:

- 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:

- 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:**

### Notes

- Used only when RTC\_ALARM is mapped on PC13
- If all RTC alternate functions are disabled and PC13MODE = 1, PC13VALUE configures the PC13 output data

### Reference Manual to LL API cross reference:

- TAFCR\_ALARMOUTTYPE 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

### Notes

- used only when RTC\_ALARM is mapped on PC13
- If all RTC alternate functions are disabled and PC13MODE = 1, PC13VALUE configures the PC13 output data

### Reference Manual to LL API cross reference:

- TAFCR ALARMOUTTYPE LL\_RTC\_GetAlarmOutputType

### LL\_RTC\_EnablePushPullMode

### Function name

**\_\_STATIC\_INLINE void LL\_RTC\_EnablePushPullMode (RTC\_TypeDef \* RTCx, uint32\_t PinMask)**

### Function description

Enable push-pull output on PC13, PC14 and/or PC15.

### Parameters

- **RTCx:** RTC Instance
- **PinMask:** This parameter can be a combination of the following values:
  - LL\_RTC\_PIN\_PC13
  - LL\_RTC\_PIN\_PC14
  - LL\_RTC\_PIN\_PC15

### Return values

- **None:**

### Notes

- PC13 forced to push-pull output if all RTC alternate functions are disabled
- PC14 and PC15 forced to push-pull output if LSE is disabled

### Reference Manual to LL API cross reference:

- TAFCR PC13MODE LL\_RTC\_EnablePushPullMode
- 
- TAFCR PC14MODE LL\_RTC\_EnablePushPullMode
- 
- TAFCR PC15MODE LL\_RTC\_EnablePushPullMode

### LL\_RTC\_DisablePushPullMode

### Function name

**\_\_STATIC\_INLINE void LL\_RTC\_DisablePushPullMode (RTC\_TypeDef \* RTCx, uint32\_t PinMask)**

### Function description

Disable push-pull output on PC13, PC14 and/or PC15.

### Parameters

- **RTCx:** RTC Instance
- **PinMask:** This parameter can be a combination of the following values:
  - LL\_RTC\_PIN\_PC13
  - LL\_RTC\_PIN\_PC14
  - LL\_RTC\_PIN\_PC15

### Return values

- **None:**

### Notes

- PC13, PC14 and/or PC15 are controlled by the GPIO configuration registers. Consequently PC13, PC14 and/or PC15 are floating in Standby mode.

### Reference Manual to LL API cross reference:

- TAFCR PC13MODE LL\_RTC\_DisablePushPullMode
- TAFCR PC14MODE LL\_RTC\_DisablePushPullMode
- TAFCR PC15MODE LL\_RTC\_DisablePushPullMode

#### LL\_RTC\_SetOutputPin

### Function name

**\_\_STATIC\_INLINE void LL\_RTC\_SetOutputPin (RTC\_TypeDef \* RTCx, uint32\_t PinMask)**

### Function description

Set PC14 and/or PC15 to high level.

### Parameters

- **RTCx:** RTC Instance
- **PinMask:** This parameter can be a combination of the following values:
  - LL\_RTC\_PIN\_PC14
  - LL\_RTC\_PIN\_PC15

### Return values

- **None:**

### Notes

- Output data configuration is possible if the LSE is disabled and PushPull output is enabled (through LL\_RTC\_EnablePushPullMode)

### Reference Manual to LL API cross reference:

- TAFCR PC14VALUE LL\_RTC\_SetOutputPin
- TAFCR PC15VALUE LL\_RTC\_SetOutputPin

#### LL\_RTC\_ResetOutputPin

### Function name

**\_\_STATIC\_INLINE void LL\_RTC\_ResetOutputPin (RTC\_TypeDef \* RTCx, uint32\_t PinMask)**

### Function description

Set PC14 and/or PC15 to low level.



### Parameters

- **RTCx:** RTC Instance
- **PinMask:** This parameter can be a combination of the following values:
  - LL\_RTC\_PIN\_PC14
  - LL\_RTC\_PIN\_PC15

### Return values

- **None:**

### Notes

- Output data configuration is possible if the LSE is disabled and PushPull output is enabled (through LL\_RTC\_EnablePushPullMode)

### Reference Manual to LL API cross reference:

- TAFCR PC14VALUE LL\_RTC\_ResetOutputPin
- TAFCR PC15VALUE LL\_RTC\_ResetOutputPin

#### 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:

- ISR 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:

- ISR 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:

- 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:

- 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:

- 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:

- 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:

- 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:**

- 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:**

- 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:**

- 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:**

- 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:**

- 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:**

- 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:**

- 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:

- WPR KEY LL\_RTC\_DisableWriteProtection

### 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:

- 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 (BYPHAD=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:**

- 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:**

- TR HT LL\_RTC\_TIME\_SetHour
- 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:**

- TR HT LL\_RTC\_TIME\_GetHour
- 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:

- TR MNT LL\_RTC\_TIME\_SetMinute
- 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 (BYP SHAD=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:

- TR MNT LL\_RTC\_TIME\_GetMinute
- 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:**

- TR ST LL\_RTC\_TIME\_SetSecond
- 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:**

- TR ST LL\_RTC\_TIME\_GetSecond
- 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:**

- TR PM LL\_RTC\_TIME\_Config
- TR HT LL\_RTC\_TIME\_Config
- TR HU LL\_RTC\_TIME\_Config
- TR MNT LL\_RTC\_TIME\_Config
- TR MNU LL\_RTC\_TIME\_Config
- TR ST LL\_RTC\_TIME\_Config
- 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 (BYPHAD=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:**

- TR HT LL\_RTC\_TIME\_Get
- TR HU LL\_RTC\_TIME\_Get
- TR MNT LL\_RTC\_TIME\_Get
- TR MNU LL\_RTC\_TIME\_Get
- TR ST LL\_RTC\_TIME\_Get
- 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:

- 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:

- 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:

- 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:

- 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:

- 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:

- 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:

- SHIFTR ADD1S LL\_RTC\_TIME\_Synchronize
- 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:

- DR YT LL\_RTC\_DATE\_SetYear
- 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 (BYPHAD=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:**

- DR YT LL\_RTC\_DATE\_GetYear
- 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:**

- 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 (BYPHAD=0), need to check if RSF flag is set before reading this bit

### Reference Manual to LL API cross reference:

- 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:

- DR MT LL\_RTC\_DATE\_SetMonth
- 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 (BYPHAD=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:

- DR MT LL\_RTC\_DATE\_GetMonth
- 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:

- DR DT LL\_RTC\_DATE\_SetDay
- 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:

- DR DT LL\_RTC\_DATE\_GetDay
- 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:**

- DR WDU LL\_RTC\_DATE\_Config
- DR MT LL\_RTC\_DATE\_Config
- DR MU LL\_RTC\_DATE\_Config
- DR DT LL\_RTC\_DATE\_Config
- DR DU LL\_RTC\_DATE\_Config
- DR YT LL\_RTC\_DATE\_Config
- 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:**

- DR WDU LL\_RTC\_DATE\_Get
- DR MT LL\_RTC\_DATE\_Get
- DR MU LL\_RTC\_DATE\_Get
- DR DT LL\_RTC\_DATE\_Get
- DR DU LL\_RTC\_DATE\_Get
- DR YT LL\_RTC\_DATE\_Get
- 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:**

- 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:

- 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:

- ALRMAR MSK4 LL\_RTC\_ALMA\_SetMask
- ALRMAR MSK3 LL\_RTC\_ALMA\_SetMask
- ALRMAR MSK2 LL\_RTC\_ALMA\_SetMask
- 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:

- ALRMAR MSK4 LL\_RTC\_ALMA\_GetMask
- ALRMAR MSK3 LL\_RTC\_ALMA\_GetMask
- ALRMAR MSK2 LL\_RTC\_ALMA\_GetMask
- 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:

- ALRMAR WDSSEL 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:

- ALRMAR WDSSEL 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:

- ALRMAR DT LL\_RTC\_ALMA\_SetDay
- 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:

- ALRMAR DT LL\_RTC\_ALMA\_GetDay
- 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:

- 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:

- 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:

- 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:

- 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:

- ALRMAR HT LL\_RTC\_ALMA\_SetHour
- 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:

- ALRMAR HT LL\_RTC\_ALMA\_GetHour
- 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:

- ALRMAR MNT LL\_RTC\_ALMA\_SetMinute
- 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:

- ALRMAR MNT LL\_RTC\_ALMA\_GetMinute
- 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:**

- ALRMAR ST `LL_RTC_ALMA_SetSecond`
- 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:**

- ALRMAR ST `LL_RTC_ALMA_GetSecond`
- 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:**

- ALRMAR PM LL\_RTC\_ALMA\_ConfigTime
- ALRMAR HT LL\_RTC\_ALMA\_ConfigTime
- ALRMAR HU LL\_RTC\_ALMA\_ConfigTime
- ALRMAR MNT LL\_RTC\_ALMA\_ConfigTime
- ALRMAR MNU LL\_RTC\_ALMA\_ConfigTime
- ALRMAR ST LL\_RTC\_ALMA\_ConfigTime
- 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:**

- ALRMAR HT LL\_RTC\_ALMA\_GetTime
- ALRMAR HU LL\_RTC\_ALMA\_GetTime
- ALRMAR MNT LL\_RTC\_ALMA\_GetTime
- ALRMAR MNU LL\_RTC\_ALMA\_GetTime
- ALRMAR ST LL\_RTC\_ALMA\_GetTime
- 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:**

- 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:

- ALRMSSR 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:

- ALRMSSR 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:

- ALRMSSR 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:

- 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:

- 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:**

- ALRMBR MSK4 LL\_RTC\_ALMB\_SetMask
- ALRMBR MSK3 LL\_RTC\_ALMB\_SetMask
- ALRMBR MSK2 LL\_RTC\_ALMB\_SetMask
- 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:**

- ALRMBR MSK4 LL\_RTC\_ALMB\_GetMask
- ALRMBR MSK3 LL\_RTC\_ALMB\_GetMask
- ALRMBR MSK2 LL\_RTC\_ALMB\_GetMask
- 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:**

- ALRMBR WSEL 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:**

- ALRMBR WDSSEL 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:**

- ALRMBR DT LL\_RTC\_ALMB\_SetDay
- 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:**

- ALRMBR DT LL\_RTC\_ALMB\_GetDay
- 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:

- ALRM BR 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:

- ALRM BR 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:

- ALRM BR 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:

- ALRM BR 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:

- ALRM BR HT LL\_RTC\_ALMB\_SetHour
- ALRM BR 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:

- ALRM BR HT LL\_RTC\_ALMB\_GetHour
- ALRM BR 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:

- ALRM BR MNT LL\_RTC\_ALMB\_SetMinute
- ALRM BR 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:**

- ALRMBR MNT LL\_RTC\_ALMB\_GetMinute
- 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:**

- ALRMBR ST LL\_RTC\_ALMB\_SetSecond
- 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:**

- ALRMBR ST LL\_RTC\_ALMB\_GetSecond
- 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:

- ALRM BR PM LL\_RTC\_ALMB\_ConfigTime
- ALRM BR HT LL\_RTC\_ALMB\_ConfigTime
- ALRM BR HU LL\_RTC\_ALMB\_ConfigTime
- ALRM BR MNT LL\_RTC\_ALMB\_ConfigTime
- ALRM BR MNU LL\_RTC\_ALMB\_ConfigTime
- ALRM BR ST LL\_RTC\_ALMB\_ConfigTime
- ALRM BR 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:

- ALRM BR HT LL\_RTC\_ALMB\_GetTime
- ALRM BR HU LL\_RTC\_ALMB\_GetTime
- ALRM BR MNT LL\_RTC\_ALMB\_GetTime
- ALRM BR MNU LL\_RTC\_ALMB\_GetTime
- ALRM BR ST LL\_RTC\_ALMB\_GetTime
- ALRM BR 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:**

- 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:**

- 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:**

- 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:**

- ALRMBSSR SS LL\_RTC\_ALMB\_GetSubSecond

**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:**

- CR TSE 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:**

- CR TSE 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:

- CR TSEDGE 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:

- CR TSEDGE 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:

- 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:

- TSTR HT LL\_RTC\_TS\_GetHour
- 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:

- TSTR MNT LL\_RTC\_TS\_GetMinute
- TSTR MNU 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:**

- TSTR ST LL\_RTC\_TS\_GetSecond
- TSTR SU 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:**

- TSTR HT LL\_RTC\_TS\_GetTime
- TSTR HU LL\_RTC\_TS\_GetTime
- TSTR MNT LL\_RTC\_TS\_GetTime
- TSTR MNU LL\_RTC\_TS\_GetTime
- TSTR ST LL\_RTC\_TS\_GetTime
- 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:**

- 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:

- TSDR MT LL\_RTC\_TS\_GetMonth
- 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:

- TSDR DT LL\_RTC\_TS\_GetDay
- 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:

- TSDR WDU LL\_RTC\_TS\_GetDate
- TSDR MT LL\_RTC\_TS\_GetDate
- TSDR MU LL\_RTC\_TS\_GetDate
- TSDR DT LL\_RTC\_TS\_GetDate
- 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:

- TSSSR 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:**

- TAFCR 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:**

- TAFCR TAMPTS LL\_RTC\_TS\_DisableOnTamper

**LL\_RTC\_TS\_SetPin**

**Function name**

`__STATIC_INLINE void LL_RTC_TS_SetPin (RTC_TypeDef * RTCx, uint32_t TSPin)`

**Function description**

Set timestamp Pin.

**Parameters**

- **RTCx:** RTC Instance
- **TSPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - LL\_RTC\_TimeStampPin\_Default: RTC\_AF1 is used as RTC TimeStamp.
  - LL\_RTC\_TimeStampPin\_Pos1: RTC\_AF2 is selected as RTC TimeStamp. (\*)
 (\*) value not defined in all devices.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- TAFCR TSINSEL LL\_RTC\_TS\_SetPin

**LL\_RTC\_TS\_GetPin**

**Function name**

`__STATIC_INLINE uint32_t LL_RTC_TS_GetPin (RTC_TypeDef * RTCx)`

**Function description**

Get timestamp Pin.

**Parameters**

- **RTCx:** RTC Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RTC\_TimeStampPin\_Default: RTC\_AF1 is used as RTC TimeStamp Pin.
  - LL\_RTC\_TimeStampPin\_Pos1: RTC\_AF2 is selected as RTC TimeStamp Pin. (\*)
 (\*) value not defined in all devices.
- **None:**

### Reference Manual to LL API cross reference:

- TAFCR TSINSEL LL\_RTC\_TS\_GetPin

### LL\_RTC\_TAMPER\_Enable

#### Function name

```
__STATIC_INLINE void LL_RTC_TAMPER_Enable (RTC_TypeDef * RTCx, uint32_t Tamper)
```

#### Function description

Enable RTC\_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 (\*)
 (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- TAFCR TAMP1E LL\_RTC\_TAMPER\_Enable
- TAFCR 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 RTC\_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 (\*)
 (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- TAFCR TAMP1E LL\_RTC\_TAMPER\_Disable
- TAFCR TAMP2E LL\_RTC\_TAMPER\_Disable
-

### 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:

- TAFCR 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:

- TAFCR 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:

- TAFCR 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:

- TAFCR 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:

- TAFCR 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:

- TAFCR 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:

- TAFCR 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:

- TAFCR 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 (\*)
 (\*) value not defined in all devices.

#### Return values

- **None:**

### Reference Manual to LL API cross reference:

- TAFCR TAMP1TRG LL\_RTC\_TAMPER\_EnableActiveLevel
- TAFCR 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 (\*)
 (\*) value not defined in all devices.

#### Return values

- **None:**



**Reference Manual to LL API cross reference:**

- TAFCR TAMP1TRG LL\_RTC\_TAMPER\_DisableActiveLevel
- TAFCR TAMP2TRG LL\_RTC\_TAMPER\_DisableActiveLevel
- 

**LL\_RTC\_TAMPER\_SetPin**
**Function name**

```
__STATIC_INLINE void LL_RTC_TAMPER_SetPin (RTC_TypeDef * RTCx, uint32_t TamperPin)
```

**Function description**

Set Tamper Pin.

**Parameters**

- **RTCx:** RTC Instance
- **TamperPin:** specifies the RTC Tamper Pin. This parameter can be one of the following values:
  - LL\_RTC\_TamperPin\_Default: RTC\_AF1 is used as RTC Tamper.
  - LL\_RTC\_TamperPin\_Pos1: RTC\_AF2 is selected as RTC Tamper. (\*)
 (\*) value not defined in all devices.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- TAFCR TAMP1INSEL LL\_RTC\_TAMPER\_SetPin

**LL\_RTC\_TAMPER\_GetPin**
**Function name**

```
__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPin (RTC_TypeDef * RTCx)
```

**Function description**

Get Tamper Pin.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_RTC\_TamperPin\_Default: RTC\_AF1 is used as RTC Tamper Pin.
  - LL\_RTC\_TamperPin\_Pos1: RTC\_AF2 is selected as RTC Tamper Pin. (\*)
 (\*) value not defined in all devices.
- **None:**

**Reference Manual to LL API cross reference:**

- TAFCR TAMP1INSEL LL\_RTC\_TAMPER\_GetPin

**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:

- 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:

- 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:

- 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\_ISR WUTWF bit = 1

### Reference Manual to LL API cross reference:

- 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:

- 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\_ISR

**Reference Manual to LL API cross reference:**

- 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:**

- WUTR WUT LL\_RTC\_WAKEUP\_GetAutoReload

**LL\_RTC\_BAK\_SetRegister**

**Function name**

`__STATIC_INLINE void LL_RTC_BAK_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)`

**Function description**

Writes a data in a 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
  - 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
- **Data:** Value between Min\_Data=0x00 and Max\_Data=0xFFFFFFFF

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- BKPxR BKP LL\_RTC\_BAK\_SetRegister

### LL\_RTC\_BAK\_GetRegister

### Function name

`__STATIC_INLINE uint32_t LL_RTC_BAK_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
  - 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

### Return values

- **Value:** between Min\_Data=0x00 and Max\_Data=0xFFFFFFFF

### Reference Manual to LL API cross reference:

- BKPxR BKP LL\_RTC\_BAK\_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:

- CR COE LL\_RTC\_CAL\_SetOutputFreq
- 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:

- CR COE LL\_RTC\_CAL\_GetOutputFreq
- CR COSEL LL\_RTC\_CAL\_GetOutputFreq

### LL\_RTC\_CAL\_EnableCoarseDigital

#### Function name

```
__STATIC_INLINE void LL_RTC_CAL_EnableCoarseDigital (RTC_TypeDef * RTCx)
```

#### Function description

Enable Coarse digital calibration.

#### 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:

- CR DCE LL\_RTC\_CAL\_EnableCoarseDigital

### LL\_RTC\_CAL\_DisableCoarseDigital

#### Function name

```
__STATIC_INLINE void LL_RTC_CAL_DisableCoarseDigital (RTC_TypeDef * RTCx)
```

#### Function description

Disable Coarse digital calibration.

#### 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:

- CR DCE LL\_RTC\_CAL\_DisableCoarseDigital

#### LL\_RTC\_CAL\_ConfigCoarseDigital

### Function name

```
__STATIC_INLINE void LL_RTC_CAL_ConfigCoarseDigital (RTC_TypeDef * RTCx, uint32_t Sign, uint32_t Value)
```

### Function description

Set the coarse digital calibration.

### Parameters

- **RTCx:** RTC Instance
- **Sign:** This parameter can be one of the following values:
  - LL\_RTC\_CALIB\_SIGN\_POSITIVE
  - LL\_RTC\_CALIB\_SIGN\_NEGATIVE
- **Value:** value of coarse calibration expressed in ppm (coded on 5 bits)

### 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)
- This Calibration value should be between 0 and 63 when using negative sign with a 2-ppm step.
- This Calibration value should be between 0 and 126 when using positive sign with a 4-ppm step.

### Reference Manual to LL API cross reference:

- CALIBR DCS LL\_RTC\_CAL\_ConfigCoarseDigital
- CALIBR DC LL\_RTC\_CAL\_ConfigCoarseDigital

#### LL\_RTC\_CAL\_GetCoarseDigitalValue

### Function name

```
__STATIC_INLINE uint32_t LL_RTC_CAL_GetCoarseDigitalValue (RTC_TypeDef * RTCx)
```

### Function description

Get the coarse digital calibration value.

### Parameters

- **RTCx:** RTC Instance

### Return values

- **value:** of coarse calibration expressed in ppm (coded on 5 bits)

### Reference Manual to LL API cross reference:

- CALIBR DC LL\_RTC\_CAL\_GetCoarseDigitalValue

#### LL\_RTC\_CAL\_GetCoarseDigitalSign

### Function name

```
__STATIC_INLINE uint32_t LL_RTC_CAL_GetCoarseDigitalSign (RTC_TypeDef * RTCx)
```



### Function description

Get the coarse digital calibration sign.

### Parameters

- **RTCx:** RTC Instance

### Return values

- **Returned:** value can be one of the following values:
  - LL\_RTC\_CALIB\_SIGN\_POSITIVE
  - LL\_RTC\_CALIB\_SIGN\_NEGATIVE

### Reference Manual to LL API cross reference:

- CALIBR DCS LL\_RTC\_CAL\_GetCoarseDigitalSign

### 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\_ISR

### Reference Manual to LL API cross reference:

- 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:

- 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\_ISR

#### Reference Manual to LL API cross reference:

- CALR CALW8 LL\_RTC\_CAL\_SetPeriod
- 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:

- CALR CALW8 LL\_RTC\_CAL\_GetPeriod
- 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\_ISR

**Reference Manual to LL API cross reference:**

- 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:**

- CALR CALM LL\_RTC\_CAL\_GetMinus

**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:**

- ISR RECALPF LL\_RTC\_IsActiveFlag\_RECALP

**LL\_RTC\_IsActiveFlag\_TAMP2**
**Function name**

```
__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (RTC_TypeDef * RTCx)
```

**Function description**

Get RTC\_TAMP2 detection flag.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- ISR TAMP2F LL\_RTC\_IsActiveFlag\_TAMP2

**LL\_RTC\_IsActiveFlag\_TAMP1**
**Function name**

```
__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (RTC_TypeDef * RTCx)
```

**Function description**

Get RTC\_TAMP1 detection flag.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- ISR TAMP1F LL\_RTC\_IsActiveFlag\_TAMP1

**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:**

- ISR 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:**

- ISR 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:

- ISR 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:

- ISR 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:

- ISR ALRAF LL\_RTC\_IsActiveFlag\_ALRA

### LL\_RTC\_ClearFlag\_TAMP2

#### Function name

```
__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)
```

#### Function description

Clear RTC\_TAMP2 detection flag.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- ISR TAMP2F LL\_RTC\_ClearFlag\_TAMP2

**LL\_RTC\_ClearFlag\_TAMP1**

**Function name**

**\_\_STATIC\_INLINE void LL\_RTC\_ClearFlag\_TAMP1 (RTC\_TypeDef \* RTCx)**

**Function description**

Clear RTC\_TAMP1 detection flag.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- ISR TAMP1F LL\_RTC\_ClearFlag\_TAMP1

**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:**

- ISR TSOVF 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:**

- ISR TSF 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:**

- ISR WUTF 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:**

- ISR ALRBF 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:**

- ISR ALRAF 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:

- ISR 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:

- ISR 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:

- ISR 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:**

- ISR 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:**

- ISR 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:**

- ISR 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:**

- ISR 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:

- ISR ALRAWF LL\_RTC\_IsActiveFlag\_ALRAW

### 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:

- CR TSIE 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:

- CR TSIE 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:

- CR WUTIE 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:

- CR WUTIE 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:**

- 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:**

- 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:**

- 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:**

- CR ALRAIE LL\_RTC\_DisableIT\_ALRA

**LL\_RTC\_EnableIT\_TAMP**
**Function name**

```
__STATIC_INLINE void LL_RTC_EnableIT_TAMP (RTC_TypeDef * RTCx)
```

**Function description**

Enable all Tamper Interrupt.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- TAFCR TAMPIE LL\_RTC\_EnableIT\_TAMP

**LL\_RTC\_DisableIT\_TAMP**
**Function name**

```
__STATIC_INLINE void LL_RTC_DisableIT_TAMP (RTC_TypeDef * RTCx)
```

**Function description**

Disable all Tamper Interrupt.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- TAFCR TAMPIE LL\_RTC\_DisableIT\_TAMP

**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:**

- 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:**

- 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:**

- 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:**

- CR ALRAIE LL\_RTC\_IsEnabledIT\_ALRA

**LL\_RTC\_IsEnabledIT\_TAMP**
**Function name**

```
__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP (RTC_TypeDef * RTCx)
```

**Function description**

Check if all the TAMPER interrupts are enabled or not.

**Parameters**

- **RTCx:** RTC Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- TAFCR TAMPIE LL\_RTC\_IsEnabledIT\_TAMP

**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 doesn't 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 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\_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.

## 89.3 RTC Firmware driver defines

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

### 89.3.1 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`, when mapped on PC13, is open-drain output

#### `LL_RTC_ALARM_OUTPUTTYPE_PUSH_PULL`

`RTC_ALARM`, when mapped on PC13, 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\_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

***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  $2^{\text{exp}11}$  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

***Coarse digital calibration sign***

LL\_RTC\_CALIB\_SIGN\_POSITIVE

Positive calibration: calendar update frequency is increased

**LL\_RTC\_CALIB\_SIGN\_NEGATIVE**

Negative calibration: calendar update frequency is decreased

**FORMAT****LL\_RTC\_FORMAT\_BIN**

Binary data format

**LL\_RTC\_FORMAT\_BCD**

BCD data format

**Get Flags Defines****LL\_RTC\_ISR\_RECALPF****LL\_RTC\_ISR\_TAMP3F****LL\_RTC\_ISR\_TAMP2F****LL\_RTC\_ISR\_TAMP1F****LL\_RTC\_ISR\_TSOVF****LL\_RTC\_ISR\_TSF****LL\_RTC\_ISR\_WUTF****LL\_RTC\_ISR\_ALRBF****LL\_RTC\_ISR\_ALRAF****LL\_RTC\_ISR\_INITF****LL\_RTC\_ISR\_RSF****LL\_RTC\_ISR\_INITS****LL\_RTC\_ISR\_SHPF****LL\_RTC\_ISR\_WUTWF****LL\_RTC\_ISR\_ALRBWF****LL\_RTC\_ISR\_ALRAWF****HOUR FORMAT****LL\_RTC\_HOURFORMAT\_24HOUR**

24 hour/day format

**LL\_RTC\_HOURFORMAT\_AMPM**

AM/PM hour format

**IT Defines****LL\_RTC\_CR\_TSIE****LL\_RTC\_CR\_WUTIE****LL\_RTC\_CR\_ALRBIE**

LL\_RTC\_CR\_ALRAIE

LL\_RTC\_TAFCR\_TAMPIE

**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)

**PIN**

LL\_RTC\_PIN\_PC13

PC13 is forced to push-pull output if all RTC alternate functions are disabled

LL\_RTC\_PIN\_PC14

PC14 is forced to push-pull output if LSE is disabled

LL\_RTC\_PIN\_PC15

PC15 is forced to push-pull output if LSE is disabled



**SHIFT SECOND**

LL\_RTC\_SHIFT\_SECOND\_DELAY

LL\_RTC\_SHIFT\_SECOND\_ADVANCE

**TAMPER1 mapping**

LL\_RTC\_TamperPin\_Default

Use RTC\_AF1 as TAMPER1

LL\_RTC\_TamperPin\_Pos1

Use RTC\_AF2 as TAMPER1

**TAMPER**

LL\_RTC\_TAMPER\_1

RTC\_TAMP1 input detection

LL\_RTC\_TAMPER\_2

RTC\_TAMP2 input detection

**TAMPER ACTIVE LEVEL**

LL\_RTC\_TAMPER\_ACTIVELEVEL\_TAMP1

RTC\_TAMP1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL\_RTC\_TAMPER\_ACTIVELEVEL\_TAMP2

RTC\_TAMP2 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.

**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.

**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 mapping**

#### LL\_RTC\_TimeStampPin\_Default

Use RTC\_AF1 as TIMESTAMP

**LL\_RTC\_TimeStampPin\_Pos1**

Use RTC\_AF2 as TIMESTAMP

***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**

Thursday

**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)

***Common Write and read registers Macros***

## `LL_RTC_WriteReg`

**Description:**

- Write a value in RTC register.

**Parameters:**

- `__INSTANCE__`: RTC Instance
- `__REG__`: Register to be written
- `__VALUE__`: Value to be written in the register

**Return value:**

- None

## `LL_RTC_ReadReg`

**Description:**

- Read a value in RTC register.

**Parameters:**

- `__INSTANCE__`: RTC Instance
- `__REG__`: Register to be read

**Return value:**

- Register: value

## 90 LL SPI Generic Driver

### 90.1 SPI Firmware driver registers structures

#### 90.1.1 LL\_SPI\_InitTypeDef

*LL\_SPI\_InitTypeDef* is defined in the `stm32f4xx_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()***.

### 90.1.2

#### **LL\_I2S\_InitTypeDef**

***LL\_I2S\_InitTypeDef*** is defined in the ***stm32f4xx\_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()***.

## 90.2

### **SPI Firmware driver API description**

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

#### 90.2.1

##### **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\_8BIT
  - LL\_SPI\_DATAWIDTH\_16BIT

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 DFF 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\_8BIT
  - LL\_SPI\_DATAWIDTH\_16BIT

#### Reference Manual to LL API cross reference:

- CR1 DFF LL\_SPI\_GetDataWidth

#### 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\_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:**

- RXCR CR 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:**

- TXCR CR 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\_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\_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\_EnabledDMAReq\_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\_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**

- **RxDData:** 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**

- **RxDData:** 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
- **TxDData:** 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
- **TxDData:** 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 output (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 output (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 output (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\_EnabledDMAReq\_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
- **TxDData**: 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).

### LL\_I2S\_InitFullDuplex

### Function name

**ErrorStatus LL\_I2S\_InitFullDuplex (SPI\_TypeDef \* I2Sxext, LL\_I2S\_InitTypeDef \* I2S\_InitStruct)**

### Function description

Configures the full duplex mode for the I2Sx peripheral using its extension I2Sxext according to the specified parameters in the I2S\_InitStruct.

### Parameters

- **I2Sxext:** SPI Instance
- **I2S\_InitStruct:** pointer to a LL\_I2S\_InitTypeDef structure

### Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2Sxext registers are Initialized
  - ERROR: I2Sxext registers are not Initialized

### Notes

- The structure pointed by I2S\_InitStruct parameter should be the same used for the master I2S peripheral. In this case, if the master is configured as transmitter, the slave will be receiver and vice versa. Or you can force a different mode by modifying the field I2S\_Mode to the value I2S\_SlaveRx or I2S\_SlaveTx independently of the master configuration.

## 90.3 SPI Firmware driver defines

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

### 90.3.1 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

#### **Datawidth**

##### LL\_SPI\_DATAWIDTH\_8BIT

Data length for SPI transfer: 8 bits

##### LL\_SPI\_DATAWIDTH\_16BIT

Data length for SPI transfer: 16 bits

#### **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

### **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

### **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

## 91 LL SYSTEM Generic Driver

### 91.1 SYSTEM Firmware driver API description

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

#### 91.1.1 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\_FSMC (\*)
  - LL\_SYSCFG\_REMAP\_FMC (\*)

###### 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\_FSMC (\*)
  - LL\_SYSCFG\_REMAP\_FMC (\*)

###### Reference Manual to LL API cross reference:

- SYSCFG\_MEMRMP MEM\_MODE LL\_SYSCFG\_GetRemapMemory

##### LL\_SYSCFG\_EnableFMCMemorySwapping

###### Function name

```
__STATIC_INLINE void LL_SYSCFG_EnableFMCMemorySwapping (void )
```

###### Function description

Enables the FMC Memory Mapping Swapping.

**Return values**

- **None:**

**Notes**

- SDRAM is accessible at 0x60000000 and NOR/RAM is accessible at 0xC0000000

**Reference Manual to LL API cross reference:**

- SYSCFG\_MEMRMP SWP\_FMC LL\_SYSCFG\_EnableFMCMemorySwapping

**LL\_SYSCFG\_DisableFMCMemorySwapping**

**Function name**

`__STATIC_INLINE void LL_SYSCFG_DisableFMCMemorySwapping (void )`

**Function description**

Disables the FMC Memory Mapping Swapping.

**Return values**

- **None:**

**Notes**

- SDRAM is accessible at 0xC0000000 (default mapping) and NOR/RAM is accessible at 0x60000000 (default mapping)

**Reference Manual to LL API cross reference:**

- SYSCFG\_MEMRMP SWP\_FMC LL\_SYSCFG\_DisableFMCMemorySwapping

**LL\_SYSCFG\_EnableCompensationCell**

**Function name**

`__STATIC_INLINE void LL_SYSCFG_EnableCompensationCell (void )`

**Function description**

Enables the Compensation cell Power Down.

**Return values**

- **None:**

**Notes**

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V

**Reference Manual to LL API cross reference:**

- SYSCFG\_CMPCR CMP\_PD LL\_SYSCFG\_EnableCompensationCell

**LL\_SYSCFG\_DisableCompensationCell**

**Function name**

`__STATIC_INLINE void LL_SYSCFG_DisableCompensationCell (void )`

**Function description**

Disables the Compensation cell Power Down.

**Return values**

- **None:**

**Notes**

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V

**Reference Manual to LL API cross reference:**

- SYSCFG\_CMPCR CMP\_PD LL\_SYSCFG\_DisableCompensationCell

**LL\_SYSCFG\_IsActiveFlag\_CMPCR**

**Function name**

`__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CMPCR (void )`

**Function description**

Get Compensation Cell ready Flag.

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- SYSCFG\_CMPCR READY LL\_SYSCFG\_IsActiveFlag\_CMPCR

**LL\_SYSCFG\_SetPHYInterface**

**Function name**

`__STATIC_INLINE void LL_SYSCFG_SetPHYInterface (uint32_t Interface)`

**Function description**

Select Ethernet PHY interface.

**Parameters**

- **Interface:** This parameter can be one of the following values:
  - LL\_SYSCFG\_PMC\_ETHMII
  - LL\_SYSCFG\_PMC\_ETHRMII

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- SYSCFG\_PMC MII\_RMII\_SEL LL\_SYSCFG\_SetPHYInterface

**LL\_SYSCFG\_GetPHYInterface**

**Function name**

`__STATIC_INLINE uint32_t LL_SYSCFG_GetPHYInterface (void )`

**Function description**

Get Ethernet PHY interface.

**Return values**

- **Returned:** value can be one of the following values:
  - LL\_SYSCFG\_PMC\_ETHMII
  - LL\_SYSCFG\_PMC\_ETHRMII
- **None:**

**Reference Manual to LL API cross reference:**

- SYSCFG\_PMC MII\_RMII\_SEL LL\_SYSCFG\_GetPHYInterface

**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 UFB\_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 UFB\_MODE LL\_SYSCFG\_GetFlashBankMode

### 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 (\*)
  - LL\_SYSCFG\_EXTI\_PORTH
 (\*) 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 (\*)
  - LL\_SYSCFG\_EXTI\_PORTH (\*) 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\_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=0xFFFF

**Notes**

- For STM32F405/407xx and STM32F415/417xx devices, the device ID is 0x413
- For STM32F42xxx and STM32F43xxx devices, the device ID is 0x419
- For STM32F401xx devices, the device ID is 0x423
- For STM32F401xx devices, the device ID is 0x433
- For STM32F411xx devices, the device ID is 0x431
- For STM32F410xx devices, the device ID is 0x458
- For STM32F412xx devices, the device ID is 0x441
- For STM32F413xx and STM32423xx devices, the device ID is 0x463
- For STM32F446xx devices, the device ID is 0x421
- For STM32F469xx and STM32F479xx devices, the device ID is 0x434

**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. For example, it is read as RevA -> 0x1000, Cat 2 revZ -> 0x1001, rev1 -> 0x1003, rev2 -> 0x1007, revY -> 0x100F for STM32F405/407xx and STM32F415/417xx devices For example, it is read as RevA -> 0x1000, Cat 2 revY -> 0x1003, rev1 -> 0x1007, rev3 -> 0x2001 for STM32F42xxx and STM32F43xxx devices For example, it is read as RevZ -> 0x1000, Cat 2 revA -> 0x1001 for STM32F401xB/C devices For example, it is read as RevA -> 0x1000, Cat 2 revZ -> 0x1001 for STM32F401xD/E devices For example, it is read as RevA -> 0x1000 for STM32F411xx, STM32F413/423xx, STM32F469/423xx, STM32F446xx and STM32F410xx devices For example, it is read as RevZ -> 0x1001, Cat 2 revB -> 0x2000, revC -> 0x3000 for STM32F412xx devices

**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\_ASYNC
  - LL\_DBGMCU\_TRACE\_SYNC\_SIZE1
  - LL\_DBGMCU\_TRACE\_SYNC\_SIZE2
  - LL\_DBGMCU\_TRACE\_SYNC\_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\_ASYNC
  - LL\_DBGMCU\_TRACE\_SYNC\_SIZE1
  - LL\_DBGMCU\_TRACE\_SYNC\_SIZE2
  - LL\_DBGMCU\_TRACE\_SYNC\_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\_TIM12\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_TIM13\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_TIM14\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_LPTIM\_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\_I2C4\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_CAN1\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_CAN2\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_CAN3\_STOP (\*)

(\*) value not defined in all devices.

## Return values

- **None:**

## Reference Manual to LL API cross reference:

- DBGMCU\_APB1\_FZ\_DBG\_TIM2\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM3\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM4\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM5\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM6\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM7\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM12\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM13\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM14\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_LPTIM\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_RTC\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_WWDG\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_IWDG\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C1\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C2\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C3\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C4\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_CAN1\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_CAN2\_STOP LL\_DBGMCU\_APB1\_GRP1\_FreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_CAN3\_STOP LL\_DBGMCU\_APB1\_GRP1\_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\_TIM12\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_TIM13\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_TIM14\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_LPTIM\_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\_I2C4\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_CAN1\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_CAN2\_STOP (\*)
  - LL\_DBGMCU\_APB1\_GRP1\_CAN3\_STOP (\*)

(\*) value not defined in all devices.

### Return values

- **None:**

**Reference Manual to LL API cross reference:**

- DBGMCU\_APB1\_FZ\_DBG\_TIM2\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM3\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM4\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM5\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM6\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM7\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM12\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM13\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_TIM14\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_LPTIM\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_RTC\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_WWDG\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_IWDG\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C1\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C2\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C3\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_I2C4\_SMBUS\_TIMEOUT LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_CAN1\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_CAN2\_STOP LL\_DBGMCU\_APB1\_GRP1\_UnFreezePeriph
- DBGMCU\_APB1\_FZ\_DBG\_CAN3\_STOP LL\_DBGMCU\_APB1\_GRP1\_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\_TIM9\_STOP (\*)
  - LL\_DBGMCU\_APB2\_GRP1\_TIM10\_STOP (\*)
  - LL\_DBGMCU\_APB2\_GRP1\_TIM11\_STOP (\*)
 (\*) value not defined in all devices.

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- DBGMCU\_APB2\_FZ\_DBG\_TIM1\_STOP LL\_DBGMCU\_APB2\_GRP1\_FreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM8\_STOP LL\_DBGMCU\_APB2\_GRP1\_FreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM9\_STOP LL\_DBGMCU\_APB2\_GRP1\_FreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM10\_STOP LL\_DBGMCU\_APB2\_GRP1\_FreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM11\_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\_TIM9\_STOP (\*)
    - LL\_DBGMCU\_APB2\_GRP1\_TIM10\_STOP (\*)
    - LL\_DBGMCU\_APB2\_GRP1\_TIM11\_STOP (\*)
- (\*) value not defined in all devices.

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- DBGMCU\_APB2\_FZ\_DBG\_TIM1\_STOP LL\_DBGMCU\_APB2\_GRP1\_UnFreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM8\_STOP LL\_DBGMCU\_APB2\_GRP1\_UnFreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM9\_STOP LL\_DBGMCU\_APB2\_GRP1\_UnFreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM10\_STOP LL\_DBGMCU\_APB2\_GRP1\_UnFreezePeriph
- DBGMCU\_APB2\_FZ\_DBG\_TIM11\_STOP LL\_DBGMCU\_APB2\_GRP1\_UnFreezePeriph

## 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

### 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

**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

## 91.2 SYSTEM Firmware driver defines

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

### 91.2.1 SYSTEM

SYSTEM

***DBGMCU APB1 GRP1 STOP IP***

**LL\_DBGMCU\_APB1\_GRP1\_TIM2\_STOP**

TIM2 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM3\_STOP**

TIM3 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM4\_STOP**

TIM4 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM5\_STOP**

TIM5 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM6\_STOP**

TIM6 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM7\_STOP**

TIM7 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM12\_STOP**

TIM12 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM13\_STOP**

TIM13 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_TIM14\_STOP**

TIM14 counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_RTC\_STOP**

RTC counter stopped when core is halted

**LL\_DBGMCU\_APB1\_GRP1\_WWDG\_STOP**

Debug Window Watchdog stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_IWDG\_STOP**

Debug Independent Watchdog stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_I2C1\_STOP**

I2C1 SMBUS timeout mode stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_I2C2\_STOP**

I2C2 SMBUS timeout mode stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_I2C3\_STOP**

I2C3 SMBUS timeout mode stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_I2C4\_STOP**

I2C4 SMBUS timeout mode stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_CAN1\_STOP**

CAN1 debug stopped when Core is halted

**LL\_DBGMCU\_APB1\_GRP1\_CAN2\_STOP**

CAN2 debug stopped when Core is halted

***DBGMCU APB2 GRP1 STOP IP***

**LL\_DBGMCU\_APB2\_GRP1\_TIM1\_STOP**

TIM1 counter stopped when core is halted

**LL\_DBGMCU\_APB2\_GRP1\_TIM8\_STOP**

TIM8 counter stopped when core is halted

**LL\_DBGMCU\_APB2\_GRP1\_TIM9\_STOP**

TIM9 counter stopped when core is halted

**LL\_DBGMCU\_APB2\_GRP1\_TIM10\_STOP**

TIM10 counter stopped when core is halted

**LL\_DBGMCU\_APB2\_GRP1\_TIM11\_STOP**

TIM11 counter stopped when core is halted

***SYSCFG BANK MODE***

**LL\_SYSCFG\_BANKMODE\_BANK1**

Flash Bank 1 base address mapped at 0x0800 0000 (AXI) and 0x0020 0000 (TCM) and Flash Bank 2 base address mapped at 0x0810 0000 (AXI) and 0x0030 0000 (TCM)

**LL\_SYSCFG\_BANKMODE\_BANK2**

Flash Bank 2 base address mapped at 0x0800 0000 (AXI) and 0x0020 0000(TCM) and Flash Bank 1 base address mapped at 0x0810 0000 (AXI) and 0x0030 0000(TCM)

***SYSCFG EXTI LINE***

**LL\_SYSCFG\_EXTI\_LINE0**

EXTI\_POSITION\_0 | EXTICR[0]

**LL\_SYSCFG\_EXTI\_LINE1**

EXTI\_POSITION\_4 | EXTICR[0]

**LL\_SYSCFG\_EXTI\_LINE2**

EXTI\_POSITION\_8 | EXTICR[0]

**LL\_SYSCFG\_EXTI\_LINE3**

EXTI\_POSITION\_12 | EXTICR[0]

**LL\_SYSCFG\_EXTI\_LINE4**

EXTI\_POSITION\_0 | EXTICR[1]

**LL\_SYSCFG\_EXTI\_LINE5**

EXTI\_POSITION\_4 | EXTICR[1]

**LL\_SYSCFG\_EXTI\_LINE6**

EXTI\_POSITION\_8 | EXTICR[1]

**LL\_SYSCFG\_EXTI\_LINE7**

EXTI\_POSITION\_12 | EXTICR[1]

**LL\_SYSCFG\_EXTI\_LINE8**

EXTI\_POSITION\_0 | EXTICR[2]

**LL\_SYSCFG\_EXTI\_LINE9**

EXTI\_POSITION\_4 | EXTICR[2]

**LL\_SYSCFG\_EXTI\_LINE10**

EXTI\_POSITION\_8 | EXTICR[2]

**LL\_SYSCFG\_EXTI\_LINE11**

EXTI\_POSITION\_12 | EXTICR[2]

**LL\_SYSCFG\_EXTI\_LINE12**

EXTI\_POSITION\_0 | EXTICR[3]

**LL\_SYSCFG\_EXTI\_LINE13**

EXTI\_POSITION\_4 | EXTICR[3]

**LL\_SYSCFG\_EXTI\_LINE14**

EXTI\_POSITION\_8 | EXTICR[3]

**LL\_SYSCFG\_EXTI\_LINE15**

EXTI\_POSITION\_12 | EXTICR[3]

***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

**LL\_SYSCFG\_EXTI\_PORTH**

EXTI PORT H

**LL\_SYSCFG\_EXTI\_PORTI**

EXTI PORT I

**LL\_SYSCFG\_EXTI\_PORTJ**

EXTI PORT J

**LL\_SYSCFG\_EXTI\_PORTK**

EXTI PORT k

***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 PMC****LL\_SYSCFG\_PMC\_ETHMII**

ETH Media MII interface

**LL\_SYSCFG\_PMC\_ETHRMII**

ETH Media RMI interface

**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\_FMC**

FMC(NOR/PSRAM 1 and 2) mapped at 0x00000000

**LL\_SYSCFG\_REMAP\_SDRAM**

FMC/SDRAM mapped at 0x00000000

**LL\_SYSCFG\_REMAP\_SRAM**

SRAM1 mapped at 0x00000000

**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

## 92 LL TIM Generic Driver

### 92.1 TIM Firmware driver registers structures

#### 92.1.1 LL\_TIM\_InitTypeDef

*LL\_TIM\_InitTypeDef* is defined in the `stm32f4xx_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()`.

#### 92.1.2 LL\_TIM\_OC\_InitTypeDef

*LL\_TIM\_OC\_InitTypeDef* is defined in the `stm32f4xx_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()`.

### 92.1.3

#### LL\_TIM\_IC\_InitTypeDef

`LL_TIM_IC_InitTypeDef` is defined in the `stm32f4xx_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()`.



**92.1.4 LL\_TIM\_ENCODER\_InitTypeDef**

*LL\_TIM\_ENCODER\_InitTypeDef* is defined in the `stm32f4xx_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()`.

**92.1.5 LL\_TIM\_HALLSENSOR\_InitTypeDef**

*LL\_TIM\_HALLSENSOR\_InitTypeDef* is defined in the `stm32f4xx_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\(\)](#).

**92.1.6**
**LL\_TIM\_BDTR\_InitTypeDef**

[LL\\_TIM\\_BDTR\\_InitTypeDef](#) is defined in the `stm32f4xx_ll_tim.h`

**Data Fields**

- ***uint32\_t OSSRState***
- ***uint32\_t OSSISate***
- ***uint32\_t LockLevel***
- ***uint8\_t DeadTime***
- ***uint16\_t BreakState***
- ***uint32\_t BreakPolarity***
- ***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::OSSISate***  
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::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.

## 92.2 TIM Firmware driver API description

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

### 92.2.1 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.

**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.

### 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

#### 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.

**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**

- 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\_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`

### 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`

### **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

### 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

### 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

### 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



## 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
- **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
- CCER CC1P LL\_TIM\_OC\_ConfigOutput
- CCER CC2P LL\_TIM\_OC\_ConfigOutput
- CCER CC3P LL\_TIM\_OC\_ConfigOutput
- CCER CC4P 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

## 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
- **Mode:** This parameter can be one of the following values:
  - LL\_TIM\_OC\_MODE\_FROZEN
  - LL\_TIM\_OC\_MODE\_ACTIVE
  - LL\_TIM\_OC\_MODE\_INACTIVE
  - LL\_TIM\_OC\_MODE\_TOGGLE
  - LL\_TIM\_OC\_MODE\_FORCED\_INACTIVE
  - LL\_TIM\_OC\_MODE\_FORCED\_ACTIVE
  - LL\_TIM\_OC\_MODE\_PWM1
  - LL\_TIM\_OC\_MODE\_PWM2

### 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

### 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

### Return values

- **Returned:** value can be one of the following values:
  - LL\_TIM\_OC\_MODE\_FROZEN
  - LL\_TIM\_OC\_MODE\_ACTIVE
  - LL\_TIM\_OC\_MODE\_INACTIVE
  - LL\_TIM\_OC\_MODE\_TOGGLE
  - LL\_TIM\_OC\_MODE\_FORCED\_INACTIVE
  - LL\_TIM\_OC\_MODE\_FORCED\_ACTIVE
  - LL\_TIM\_OC\_MODE\_PWM1
  - LL\_TIM\_OC\_MODE\_PWM2

**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

**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
- **Polarity:** This parameter can be one of the following values:
  - LL\_TIM\_OC\_POLARITY\_HIGH
  - LL\_TIM\_OC\_POLARITY\_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

**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

### Return values

- **Returned:** value can be one of the following values:
  - LL\_TIM\_OC\_POLARITY\_HIGH
  - LL\_TIM\_OC\_POLARITY\_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

### 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
- **IdleState:** This parameter can be one of the following values:
  - LL\_TIM\_OC\_IDLESTATE\_LOW
  - LL\_TIM\_OC\_IDLESTATE\_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\_OIS1N\_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

### 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

#### 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\_OIS1N\_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

### 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

### 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

#### 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

### 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

#### 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

**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
- 

**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

**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

**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

### 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

#### 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

### 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
- 

#### 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

### 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

#### 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

### 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

#### 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

### 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
- 

### 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.

### 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.

### 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.

**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.

**Reference Manual to LL API cross reference:**

- CCR4 CCR4 LL\_TIM\_OC\_SetCompareCH4

**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.

**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.

**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.

**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.

### Reference Manual to LL API cross reference:

- CCR4 CCR4 LL\_TIM\_OC\_GetCompareCH4

## 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 connected 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.

### 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.

### 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.

**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.

**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

### 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

### 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\_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

### 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

### 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\_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)`

### 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`

### 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 BKP LL_TIM_ConfigBRK`

### **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\_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
- **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

## 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\_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 OR 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\_OR ITR2\_RMP LL\_TIM\_SetRemap
- TIM2\_OR ITR1\_RMP LL\_TIM\_SetRemap
- TIM5\_OR ITR1\_RMP LL\_TIM\_SetRemap
- TIM5\_OR TI4\_RMP LL\_TIM\_SetRemap
- TIM9\_OR ITR1\_RMP LL\_TIM\_SetRemap
- TIM11\_OR TI1\_RMP LL\_TIM\_SetRemap
- LPTIM1\_OR OR LL\_TIM\_SetRemap

## 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\_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\_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\_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\_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\_EnableDMARReq\_TRIG

#### Function name

```
__STATIC_INLINE void LL_TIM_EnableDMARReq_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\_EnableDMARReq\_TRIG

### LL\_TIM\_DisableDMARReq\_TRIG

#### Function name

```
__STATIC_INLINE void LL_TIM_DisableDMARReq_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\_DisableDMARReq\_TRIG

### LL\_TIM\_IsEnabledDMARReq\_TRIG

#### Function name

```
__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMARReq_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\_IsEnabledDMARReq\_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\_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
- **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 AOE, BKP, BKE, OSSR, OSSI 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.

## 92.3 TIM Firmware driver defines

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

### 92.3.1 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

**Break Enable**

#### LL\_TIM\_BREAK\_DISABLE

Break function disabled

#### LL\_TIM\_BREAK\_ENABLE

Break function enabled

**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

**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\_DOWN

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\_UP

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

***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 8 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

***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

***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

**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\_COMIF**

COM interrupt flag

**LL\_TIM\_SR\_TIF**

Trigger interrupt flag

**LL\_TIM\_SR\_BIF**

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

**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 TlxFP1 rising edge, TlxFP1 is not inverted

**LL\_TIM\_IC\_POLARITY\_FALLING**

The circuit is sensitive to TlxFP1 falling edge, TlxFP1 is inverted

**LL\_TIM\_IC\_POLARITY\_BOTHEDGE**

The circuit is sensitive to both TlxFP1 rising and falling edges, TlxFP1 is not inverted

***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

**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

**Output Configuration Polarity**

#### LL\_TIM\_OCPOLARITY\_HIGH

OCxactive high

#### LL\_TIM\_OCPOLARITY\_LOW

OCxactive low

**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, OCx/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  $CCxNE=1$

**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

**TIM11 External Input Capture 1 Remap**

**LL\_TIM\_TIM11\_TI1\_RMP\_GPIO**

TIM11 channel 1 is connected to GPIO

**LL\_TIM\_TIM11\_TI1\_RMP\_GPIO1**

TIM11 channel 1 is connected to GPIO

**LL\_TIM\_TIM11\_TI1\_RMP\_GPIO2**

TIM11 channel 1 is connected to GPIO

**LL\_TIM\_TIM11\_TI1\_RMP\_HSE\_RTC**

TIM11 channel 1 is connected to HSE\_RTC

***TIM2 Internal Trigger1 Remap TIM8***

**LL\_TIM\_TIM2\_ITR1\_RMP\_TIM8\_TRGO**

TIM2\_ITR1 is connected to TIM8\_TRGO

**LL\_TIM\_TIM2\_ITR1\_RMP\_ETH\_PTP**

TIM2\_ITR1 is connected to ETH\_PTP

**LL\_TIM\_TIM2\_ITR1\_RMP\_OTG\_FS\_SOF**

TIM2\_ITR1 is connected to OTG\_FS SOF

**LL\_TIM\_TIM2\_ITR1\_RMP\_OTG\_HS\_SOF**

TIM2\_ITR1 is connected to OTG\_HS SOF

***TIM5 External Input Ch4 Remap***

**LL\_TIM\_TIM5\_TI4\_RMP\_GPIO**

TIM5 channel 4 is connected to GPIO

**LL\_TIM\_TIM5\_TI4\_RMP\_LSI**

TIM5 channel 4 is connected to LSI internal clock

**LL\_TIM\_TIM5\_TI4\_RMP\_LSE**

TIM5 channel 4 is connected to LSE

**LL\_TIM\_TIM5\_TI4\_RMP\_RTC**

TIM5 channel 4 is connected to RTC wakeup interrupt

***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

#### **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 (TI2FP2) is used as trigger input

#### LL\_TIM\_TS\_ETRF

Filtered external Trigger (ETRF) 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\_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\_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\_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\_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

## 93 LL USART Generic Driver

### 93.1 USART Firmware driver registers structures

#### 93.1.1 LL\_USART\_InitTypeDef

*LL\_USART\_InitTypeDef* is defined in the `stm32f4xx_ll_usart.h`

##### Data Fields

- *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::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()`.

#### 93.1.2 LL\_USART\_ClockInitTypeDef

*LL\_USART\_ClockInitTypeDef* is defined in the `stm32f4xx_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.

## 93.2 USART Firmware driver API description

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

### 93.2.1 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\_SR 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\_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\_8B
  - LL\_USART\_DATAWIDTH\_9B

### Return values

- **None:**

### Reference Manual to LL API cross reference:

- CR1 M 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\_8B
  - LL\_USART\_DATAWIDTH\_9B

### Reference Manual to LL API cross reference:

- CR1 M LL\_USART\_GetDataWidth

### 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\_GetLastCikPulseOutput

### 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\_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\_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 M LL\_USART\_ConfigCharacter
- CR2 STOP LL\_USART\_ConfigCharacter

**LL\_USART\_SetNodeAddress**
**Function name**

```
__STATIC_INLINE void LL_USART_SetNodeAddress (USART_TypeDef * USARTx, uint32_t NodeAddress)
```

**Function description**

Set Address of the USART node.

**Parameters**

- **USARTx:** USART Instance
- **NodeAddress:** 4 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.

**Reference Manual to LL API cross reference:**

- CR2 ADD LL\_USART\_SetNodeAddress

**LL\_USART\_GetNodeAddress**
**Function name**

```
__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (USART_TypeDef * USARTx)
```

**Function description**

Return 4 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**

- only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant)

**Reference Manual to LL API cross reference:**

- CR2 ADD LL\_USART\_GetNodeAddress

**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\_SetBaudRate

### Function name

```
__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, 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
- **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)

### 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 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
- **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.

**Reference Manual to LL API cross reference:**

- BRR BRR LL\_USART\_GetBaudRate

**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\_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 HDSSEL 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_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\_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()` function Clear CLKEN in CR2 using `LL_USART_DisableSCLKOutput()` function Clear SCEN in CR3 using `LL_USART_DisableSmartcard()` function Clear IREN in CR3 using `LL_USART_DisableIrda()` function Clear 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() function Clear IREN in CR3 using LL\_USART\_DisableIrda() function Clear SCEN in CR3 using LL\_USART\_DisableSmartcard() function Clear HDSEL in CR3 using LL\_USART\_DisableHalfDuplex() function Set 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() function Clear IREN in CR3 using LL\_USART\_DisableIrda() function Clear HDSEL in CR3 using LL\_USART\_DisableHalfDuplex() function Configure STOP in CR2 using LL\_USART\_SetStopBitsLength() function Set CLKEN in CR2 using LL\_USART\_EnableSCLKOutput() function Set 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() function Clear CLKEN in CR2 using LL\_USART\_DisableSCLKOutput() function Clear SCEN in CR3 using LL\_USART\_DisableSmartcard() function Clear HDSEL in CR3 using LL\_USART\_DisableHalfDuplex() function Configure STOP in CR2 using LL\_USART\_SetStopBitsLength() function Set 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() function Clear CLKEN in CR2 using LL\_USART\_DisableSCLKOutput() function Clear SCEN in CR3 using LL\_USART\_DisableSmartcard() function Clear IREN in CR3 using LL\_USART\_DisableIrda() function Clear 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:

- SR 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:

- SR 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:

- SR NF 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:**

- SR 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:**

- SR IDLE LL\_USART\_IsActiveFlag\_IDLE

**LL\_USART\_IsActiveFlag\_RXNE**

**Function name**

**\_\_STATIC\_INLINE uint32\_t LL\_USART\_IsActiveFlag\_RXNE (USART\_TypeDef \* USARTx)**

**Function description**

Check if the USART Read Data Register Not Empty Flag is set or not.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- SR RXNE LL\_USART\_IsActiveFlag\_RXNE

**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:**

- SR TC LL\_USART\_IsActiveFlag\_TC

**LL\_USART\_IsActiveFlag\_TXE**
**Function name**

```
__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE (USART_TypeDef * USARTx)
```

**Function description**

Check if the USART Transmit Data Register Empty Flag is set or not.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **State:** of bit (1 or 0).

**Reference Manual to LL API cross reference:**

- SR TXE LL\_USART\_IsActiveFlag\_TXE

**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:**

- SR LBD 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 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:**

- SR CTS LL\_USART\_IsActiveFlag\_nCTS

**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:**

- CR1 SBK 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:**

- CR1 RWU LL\_USART\_IsActiveFlag\_RWU

**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:**

**Notes**

- Clearing this flag is done by a read access to the USARTx\_SR register followed by a read access to the USARTx\_DR register.
- Please also consider that when clearing this flag, other flags as NE, FE, ORE, IDLE would also be cleared.

**Reference Manual to LL API cross reference:**

- SR PE 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:**

#### Notes

- Clearing this flag is done by a read access to the USARTx\_SR register followed by a read access to the USARTx\_DR register.
- Please also consider that when clearing this flag, other flags as PE, NE, ORE, IDLE would also be cleared.

#### Reference Manual to LL API cross reference:

- SR FE LL\_USART\_ClearFlag\_FE

### LL\_USART\_ClearFlag\_NE

#### Function name

```
__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)
```

#### Function description

Clear Noise detected Flag.

#### Parameters

- **USARTx:** USART Instance

#### Return values

- **None:**

#### Notes

- Clearing this flag is done by a read access to the USARTx\_SR register followed by a read access to the USARTx\_DR register.
- Please also consider that when clearing this flag, other flags as PE, FE, ORE, IDLE would also be cleared.

#### Reference Manual to LL API cross reference:

- SR NF 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:**



**Notes**

- Clearing this flag is done by a read access to the USARTx\_SR register followed by a read access to the USARTx\_DR register.
- Please also consider that when clearing this flag, other flags as PE, NE, FE, IDLE would also be cleared.

**Reference Manual to LL API cross reference:**

- SR ORE 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:**

**Notes**

- Clearing this flag is done by a read access to the USARTx\_SR register followed by a read access to the USARTx\_DR register.
- Please also consider that when clearing this flag, other flags as PE, NE, FE, ORE would also be cleared.

**Reference Manual to LL API cross reference:**

- SR IDLE LL\_USART\_ClearFlag\_IDLE

**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:**

- SR TC LL\_USART\_ClearFlag\_TC

**LL\_USART\_ClearFlag\_RXNE**
**Function name**

```
__STATIC_INLINE void LL_USART_ClearFlag_RXNE (USART_TypeDef * USARTx)
```

**Function description**

Clear RX Not Empty Flag.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- SR RXNE LL\_USART\_ClearFlag\_RXNE

**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:**

- SR LBD 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:**

- SR CTS LL\_USART\_ClearFlag\_nCTS

**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**
**Function name**

```
__STATIC_INLINE void LL_USART_EnableIT_RXNE (USART_TypeDef * USARTx)
```

**Function description**

Enable RX Not Empty Interrupt.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 RXNEIE LL\_USART\_EnableIT\_RXNE

**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**
**Function name**

```
__STATIC_INLINE void LL_USART_EnableIT_TXE (USART_TypeDef * USARTx)
```

**Function description**

Enable TX Empty Interrupt.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 TXEIE LL\_USART\_EnableIT\_TXE

### 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\_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\_SR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx\_SR 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\_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**
**Function name**

```
__STATIC_INLINE void LL_USART_DisableIT_RXNE (USART_TypeDef * USARTx)
```

**Function description**

Disable RX Not Empty Interrupt.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 RXNEIE LL\_USART\_DisableIT\_RXNE

### 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

#### Function name

```
__STATIC_INLINE void LL_USART_DisableIT_TXE (USART_TypeDef * USARTx)
```

#### Function description

Disable TX Empty Interrupt.

#### Parameters

- **USARTx:** USART Instance

#### Return values

- **None:**

#### Reference Manual to LL API cross reference:

- CR1 TXEIE LL\_USART\_DisableIT\_TXE

### 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\_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\_SR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx\_SR 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\_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

#### Function name

```
__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE (USART_TypeDef * USARTx)
```

#### Function description

Check if the USART RX Not Empty 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 RXNEIE LL\_USART\_IsEnabledIT\_RXNE

### 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

#### Function name

```
__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE (USART_TypeDef * USARTx)
```

#### Function description

Check if the USART TX Empty 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 TXEIE LL\_USART\_IsEnabledIT\_TXE

**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\_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\_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\_DMA\_GetRegAddr

### Function name

```
__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (USART_TypeDef * USARTx)
```

### Function description

Get the data register address used for DMA transfer.

### Parameters

- **USARTx:** USART Instance

### Return values

- **Address:** of data register

### Notes

- Address of Data Register is valid for both Transmit and Receive transfers.

### Reference Manual to LL API cross reference:

- DR DR 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:

- DR DR 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:**

- DR DR 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:**

- DR DR 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:**

- DR DR LL\_USART\_TransmitData9

**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:**

- CR1 SBK LL\_USART\_RequestBreakSending

**LL\_USART\_RequestEnterMuteMode**
**Function name**

```
__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)
```

**Function description**

Put USART in Mute mode.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 RWU LL\_USART\_RequestEnterMuteMode

**LL\_USART\_RequestExitMuteMode**
**Function name**

```
__STATIC_INLINE void LL_USART_RequestExitMuteMode (USART_TypeDef * USARTx)
```

**Function description**

Put USART in Active mode.

**Parameters**

- **USARTx:** USART Instance

**Return values**

- **None:**

**Reference Manual to LL API cross reference:**

- CR1 RWU LL\_USART\_RequestExitMuteMode

**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 IP 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 IP 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:**

## 93.3 USART Firmware driver defines

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

### 93.3.1 USART

USART

#### ***Clock Signal***

#### LL\_USART\_CLOCK\_DISABLE

Clock signal not provided

#### LL\_USART\_CLOCK\_ENABLE

Clock signal provided

#### ***Datawidth***

#### 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

#### ***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

#### ***Get Flags Defines***

#### LL\_USART\_SR\_PE

Parity error flag

#### LL\_USART\_SR\_FE

Framing error flag



**LL\_USART\_SR\_NE**

Noise detected flag

**LL\_USART\_SR\_ORE**

Overrun error flag

**LL\_USART\_SR\_IDLE**

Idle line detected flag

**LL\_USART\_SR\_RXNE**

Read data register not empty flag

**LL\_USART\_SR\_TC**

Transmission complete flag

**LL\_USART\_SR\_TXE**

Transmit data register empty flag

**LL\_USART\_SR\_LBD**

LIN break detection flag

**LL\_USART\_SR\_CTS**

CTS 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**

Read data register not empty interrupt enable

**LL\_USART\_CR1\_TCIE**

Transmission complete interrupt enable

**LL\_USART\_CR1\_TXEIE**

Transmit data register empty interrupt enable

**LL\_USART\_CR1\_PEIE**

Parity error

**LL\_USART\_CR2\_LBDIE**

LIN break detection interrupt enable

**LL\_USART\_CR3\_EIE**

Error interrupt enable

**LL\_USART\_CR3\_CTSIE**

CTS 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

***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

**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

**Exported\_Macros\_Helper**

**\_\_LL\_USART\_DIV\_SAMPLING8\_100**

**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
- **\_\_BAUDRATE\_\_**: Baud rate value to achieve

**Return value:**

- USARTDIV: value to be used for BRR register filling in OverSampling\_8 case

**\_\_LL\_USART\_DIVMANT\_SAMPLING8**

**\_\_LL\_USART\_DIVFRAQ\_SAMPLING8**

**\_\_LL\_USART\_DIV\_SAMPLING8**

**\_\_LL\_USART\_DIV\_SAMPLING16\_100**

**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
- **\_\_BAUDRATE\_\_**: Baud rate value to achieve

**Return value:**

- USARTDIV: value to be used for BRR register filling in OverSampling\_16 case

**\_\_LL\_USART\_DIVMANT\_SAMPLING16**

**\_\_LL\_USART\_DIVFRAQ\_SAMPLING16**

**\_\_LL\_USART\_DIV\_SAMPLING16**

**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

## 94 LL UTILS Generic Driver

### 94.1 UTILS Firmware driver registers structures

#### 94.1.1 LL\_UTILS\_PLLInitTypeDef

*LL\_UTILS\_PLLInitTypeDef* is defined in the `stm32f4xx_ll_utils.h`

##### Data Fields

- *uint32\_t PLLM*
- *uint32\_t PLLN*
- *uint32\_t PLLP*

##### 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 = RCC\_PLLN\_MIN\_VALUE* and *Max\_Data = RCC\_PLLN\_MIN\_VALUE*This feature can be modified afterwards using unitary function *LL\_RCC\_PLL\_ConfigDomain\_SYS()*.
- *uint32\_t LL\_UTILS\_PLLInitTypeDef::PLLP*  
Division for the main system clock. This parameter can be a value of *RCC\_LL\_EC\_PLLP\_DIV*This feature can be modified afterwards using unitary function *LL\_RCC\_PLL\_ConfigDomain\_SYS()*.

#### 94.1.2 LL\_UTILS\_ClkInitTypeDef

*LL\_UTILS\_ClkInitTypeDef* is defined in the `stm32f4xx_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()*.

### 94.2 UTILS Firmware driver API description

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

#### 94.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 180000000 Hz.

This section contains the following APIs:

- *LL\_SetSystemCoreClock()*

- `LL_PLL_ConfigSystemClock_HSI()`
- `LL_PLL_ConfigSystemClock_HSE()`

### 94.2.2 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]:**

#### 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]:**

#### 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]:**

#### 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\_WLCSP36\_UFQFPN48\_LQFP64 (\*)
    - LL\_UTILS\_PACKAGETYPE\_WLCSP168\_FBGA169\_LQFP100\_LQFP64\_UFQFPN48 (\*)
    - LL\_UTILS\_PACKAGETYPE\_WLCSP64\_WLCSP81\_LQFP176\_UFBGA176 (\*)
    - LL\_UTILS\_PACKAGETYPE\_LQFP144\_UFBGA144\_UFBGA144\_UFBGA100 (\*)
    - LL\_UTILS\_PACKAGETYPE\_LQFP100\_LQFP208\_TFBGA216 (\*)
    - LL\_UTILS\_PACKAGETYPE\_LQFP208\_TFBGA216 (\*)
    - LL\_UTILS\_PACKAGETYPE\_TQFP64\_UFBGA144\_LQFP144 (\*)
- (\*) value not defined in all devices.

### 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\_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) / PLLP) PLLM: ensure that the VCO input frequency ranges from RCC\_PLLVCO\_INPUT\_MIN to RCC\_PLLVCO\_INPUT\_MAX (PLLVCO\_input = HSI frequency / PLLM) PLLN: ensure that the VCO output frequency is between RCC\_PLLVCO\_OUTPUT\_MIN and RCC\_PLLVCO\_OUTPUT\_MAX (PLLVCO\_output = PLLVCO\_input \* PLLN) PLLP: ensure that max frequency at 180000000 Hz is reach (PLLVCO\_output / PLLP)

**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 = 26000000
- **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. PLL output frequency = (((HSI frequency / PLLM) \* PLLN) / PLLP) PLLM: ensure that the VCO input frequency ranges from RCC\_PLLVCO\_INPUT\_MIN to RCC\_PLLVCO\_INPUT\_MAX (PLLVCO\_input = HSI frequency / PLLM) PLLN: ensure that the VCO output frequency is between RCC\_PLLVCO\_OUTPUT\_MIN and RCC\_PLLVCO\_OUTPUT\_MAX (PLLVCO\_output = PLLVCO\_input \* PLLN) PLLP: ensure that max frequency at 180000000 Hz is reach (PLLVCO\_output / PLLP)

## 94.3 UTILS Firmware driver defines

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

### 94.3.1 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\_WLCSP36\_UFQFPN48\_LQFP64**

WLCSP36 or UFQFPN48 or LQFP64 package type

**LL\_UTILS\_PACKAGETYPE\_WLCSP168\_FBGA169\_LQFP100\_LQFP64\_UFQFPN48**

WLCSP168 or FBGA169 or LQFP100 or LQFP64 or UFQFPN48 package type

**LL\_UTILS\_PACKAGETYPE\_WLCSP64\_WLCSP81\_LQFP176\_UFBGA176**

WLCSP64 or WLCSP81 or LQFP176 or UFBGA176 package type

**LL\_UTILS\_PACKAGETYPE\_LQFP144\_UFBGA144\_UFBGA144\_UFBGA100**

LQFP144 or UFBGA144 or UFBGA144 or UFBGA100 package type

**LL\_UTILS\_PACKAGETYPE\_LQFP100\_LQFP208\_TFBGA216**

LQFP100 or LQFP208 or TFBGA216 package type

**LL\_UTILS\_PACKAGETYPE\_LQFP208\_TFBGA216**

LQFP208 or TFBGA216 package type

**LL\_UTILS\_PACKAGETYPE\_TQFP64\_UFBGA144\_LQFP144**

TQFP64 or UFBGA144 or LQFP144 package type

## 95 LL WWDG Generic Driver

### 95.1 WWDG Firmware driver API description

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

#### 95.1.1 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

**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 WDG TB 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

**Reference Manual to LL API cross reference:**

- CFR WDG TB 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 than 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

## 95.2 WWDG Firmware driver defines

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

### 95.2.1 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

**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



## 96 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 STM32F4 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: stm32f4xx\_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 (stm32f4xx\_hal\_conf\_template.c).

#### Which header files should I include in my application to use the HAL drivers?

Only stm32f4xx\_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 (stm32f4xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f4xx\_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\_stm32f4xx.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 functions 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 stm32f4xx\_hal\_msp.c. A template is provided in the HAL driver folders (stm32f4xx\_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 SysTick 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() instm32f4xx\_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 stm32f4xx\_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 stm32f4xx\_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 stm32f4xx\_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	Revision	Changes
05-May-2014	1	Initial release.
03-Apr-2015	2	<p>Added CEC, FMPI2C, QSPI and SPDIFRX in Table 1.</p> <p>Added STM32F446xx, cec, dcmi, fmpi2c, fmpi2c_ex, spdifrx and qspi in Table 1.</p> <p>Updated Common macros section in HAL common resources.</p> <p>Added HAL CEC Generic Driver, HAL FLASH__RAMFUNC Generic Driver HAL FMPI2C Generic Driver, HAL FMPI2C Extension Driver, HAL QSPI Generic Driver and HAL SPDIFRX Generic Driver.</p>
15-Sep-2015	3	<p>Added DSI and LPTIM and removed msp_template in Table 1.</p> <p>Added STM32F469xx, STM32F479xx, STM32F410xx, dsi, ltdc_ex and lptim in Table 1.</p> <p>Added HAL DSI Generic Driver, HAL LPTIM Generic Driver and HAL LTDC Generic Driver.</p>
02-Sep-2016	4	<p>Added DFSDM in Acronyms and definitions.</p> <p>Added STM32F412Cx, STM32F412Rx, STM32F412Vx, STM32F412Zx and DFSDM in Table 1.</p> <p>Added HAL DFSDM Generic Driver.</p>
20-Feb-2017	5	<p>Added MMC in <a href="#">Section 2 Acronyms and definitions</a>.</p> <p>Added STM32F413xx, STM32F423xx, MMC row and LL driver rows in Table : "List of devices supported by HAL drivers".</p> <p>Added Section 43: "HAL MMC Generic Driver" description.</p> <p>Added description of LL Generic drivers.</p> <p>Updated FAQ section.</p>
31-Jul-2020	6	<p>Updated <a href="#">Section Introduction</a>.</p> <p>Added <a href="#">Section 1 General information</a>.</p> <p>Updated <a href="#">Section 2 Acronyms and definitions</a>.</p> <p>Updated <a href="#">Figure 1. Example of project template</a>.</p> <p>Added <a href="#">Section 4 Overview of low-layer drivers</a> and <a href="#">Section 5 Cohabiting of HAL and LL</a>.</p> <p>Updated HAL drivers.</p> <p>Added low-layer drivers.</p>
18-Jun-2021	7	<p><a href="#">Section 40 HAL IRDA Generic Driver</a>: updated note at the end of <a href="#">Section 40.2.2 Callback registration</a>.</p> <p><a href="#">Section 70 HAL UART Generic Driver</a>:</p> <ul style="list-style-type: none"> <li>Added support for the following new APIs: HAL_UARTEEx_ReceiveToldle(), HAL_UARTEEx_ReceiveToldle_IT() and HAL_UARTEEx_ReceiveToldle_DMA().</li> <li>Updated note at the end of <a href="#">Section 70.2.2 Callback registration</a>.</li> </ul> <p><a href="#">Section 71 HAL USART Generic Driver</a>: updated note at the end of <a href="#">Section 71.2.2 Callback registration</a>.</p>

## Contents

<b>1</b>	<b>General information</b>	<b>3</b>
<b>2</b>	<b>Acronyms and definitions</b>	<b>4</b>
<b>3</b>	<b>Overview of HAL drivers</b>	<b>7</b>
3.1	HAL and user-application files	8
3.1.1	HAL driver files	8
3.1.2	User-application files	8
3.2	HAL data structures	10
3.2.1	Peripheral handle structures	10
3.2.2	Initialization and configuration structure	11
3.2.3	Specific process structures	12
3.3	API classification	13
3.4	Devices supported by HAL drivers	14
3.5	HAL driver rules	16
3.5.1	HAL API naming rules	16
3.5.2	HAL general naming rules	17
3.5.3	HAL interrupt handler and callback functions	18
3.6	HAL generic APIs	18
3.7	HAL extension APIs	19
3.7.1	HAL extension model overview	19
3.7.2	HAL extension model cases	20
3.8	File inclusion model	23
3.9	HAL common resources	24
3.10	HAL configuration	24
3.11	HAL system peripheral handling	25
3.11.1	Clocks	25
3.11.2	GPIOs	26
3.11.3	Cortex <sup>®</sup> NVIC and SysTick timer	27
3.11.4	PWR	27
3.11.5	EXTI	28
3.11.6	DMA	29
3.12	How to use HAL drivers	30
3.12.1	HAL usage models	30
3.12.2	HAL initialization	30
3.12.3	HAL I/O operation process	32
3.12.4	Timeout and error management	35

<b>4</b>	<b>Overview of low-layer drivers</b>	<b>39</b>
4.1	Low-layer files	39
4.2	Overview of low-layer APIs and naming rules	40
4.2.1	Peripheral initialization functions	40
4.2.2	Peripheral register-level configuration functions	42
<b>5</b>	<b>Cohabiting of HAL and LL</b>	<b>44</b>
5.1	Low-layer driver used in Standalone mode	44
5.2	Mixed use of low-layer APIs and HAL drivers	44
<b>6</b>	<b>HAL System Driver</b>	<b>45</b>
6.1	HAL Firmware driver API description	45
6.1.1	How to use this driver	45
6.1.2	Initialization and Configuration functions	45
6.1.3	HAL Control functions	45
6.1.4	Detailed description of functions	46
6.2	HAL Firmware driver defines	53
6.2.1	HAL	53
<b>7</b>	<b>HAL ADC Generic Driver</b>	<b>56</b>
7.1	ADC Firmware driver registers structures	56
7.1.1	ADC_InitTypeDef	56
7.1.2	ADC_ChannelConfTypeDef	57
7.1.3	ADC_AnalogWDGConfTypeDef	58
7.1.4	ADC_HandleTypeDef	58
7.2	ADC Firmware driver API description	59
7.2.1	ADC Peripheral features	59
7.2.2	How to use this driver	59
7.2.3	Initialization and de-initialization functions	62
7.2.4	IO operation functions	62
7.2.5	Peripheral Control functions	63
7.2.6	Peripheral State and errors functions	63
7.2.7	Detailed description of functions	63
7.3	ADC Firmware driver defines	70
7.3.1	ADC	70
<b>8</b>	<b>HAL ADC Extension Driver</b>	<b>78</b>
8.1	ADCEX Firmware driver registers structures	78
8.1.1	ADC_InjectionConfTypeDef	78
8.1.2	ADC_MultiModeTypeDef	79
8.2	ADCEX Firmware driver API description	79

8.2.1	How to use this driver	80
8.2.2	Extended features functions	80
8.2.3	Detailed description of functions	81
8.3	ADCEX Firmware driver defines	85
8.3.1	ADCEX	85
<b>9</b>	<b>HAL CAN Generic Driver</b>	<b>87</b>
9.1	CAN Firmware driver registers structures	87
9.1.1	CAN_InitTypeDef	87
9.1.2	CAN_FilterTypeDef	87
9.1.3	CAN_TxHeaderTypeDef	88
9.1.4	CAN_RxHeaderTypeDef	89
9.1.5	__CAN_HandleTypeDef	90
9.2	CAN Firmware driver API description	90
9.2.1	How to use this driver	90
9.2.2	Initialization and de-initialization functions	92
9.2.3	Configuration functions	92
9.2.4	Control functions	92
9.2.5	Interrupts management	92
9.2.6	Peripheral State and Error functions	93
9.2.7	Detailed description of functions	93
9.3	CAN Firmware driver defines	103
9.3.1	CAN	103
<b>10</b>	<b>HAL CEC Generic Driver</b>	<b>112</b>
10.1	CEC Firmware driver registers structures	112
10.1.1	CEC_InitTypeDef	112
10.1.2	CEC_HandleTypeDef	113
10.2	CEC Firmware driver API description	113
10.2.1	How to use this driver	113
10.2.2	Initialization and Configuration functions	114
10.2.3	IO operation functions	114
10.2.4	Peripheral Control function	114
10.2.5	Detailed description of functions	115
10.3	CEC Firmware driver defines	118
10.3.1	CEC	118
<b>11</b>	<b>HAL CORTEX Generic Driver</b>	<b>127</b>
11.1	CORTEX Firmware driver registers structures	127
11.1.1	MPU_Region_InitTypeDef	127

11.2	CORTEX Firmware driver API description .....	128
11.2.1	How to use this driver .....	128
11.2.2	Initialization and de-initialization functions .....	128
11.2.3	Peripheral Control functions .....	129
11.2.4	Detailed description of functions .....	129
11.3	CORTEX Firmware driver defines .....	134
11.3.1	CORTEX .....	134
<b>12</b>	<b>HAL CRC Generic Driver .....</b>	<b>138</b>
12.1	CRC Firmware driver registers structures .....	138
12.1.1	CRC_HandleTypeDef .....	138
12.2	CRC Firmware driver API description .....	138
12.2.1	How to use this driver .....	138
12.2.2	Initialization and de-initialization functions .....	138
12.2.3	Peripheral Control functions .....	138
12.2.4	Peripheral State functions .....	139
12.2.5	Detailed description of functions .....	139
12.3	CRC Firmware driver defines .....	141
12.3.1	CRC .....	141
<b>13</b>	<b>HAL CRYPT Generic Driver .....</b>	<b>142</b>
13.1	CRYPT Firmware driver registers structures .....	142
13.1.1	CRYPT_ConfigTypeDef .....	142
13.1.2	__CRYPT_HandleTypeDef .....	142
13.2	CRYPT Firmware driver API description .....	144
13.2.1	How to use this driver .....	144
13.2.2	Initialization, de-initialization and Set and Get configuration functions .....	146
13.2.3	Encrypt Decrypt functions .....	146
13.2.4	CRYPT IRQ handler management .....	147
13.2.5	Detailed description of functions .....	147
13.3	CRYPT Firmware driver defines .....	152
13.3.1	CRYPT .....	153
<b>14</b>	<b>HAL CRYPT Extension Driver .....</b>	<b>158</b>
14.1	CRYPEx Firmware driver API description .....	158
14.1.1	How to use this driver .....	158
14.1.2	Extended AES processing functions .....	158
14.1.3	Detailed description of functions .....	158
<b>15</b>	<b>HAL DAC Generic Driver .....</b>	<b>160</b>
15.1	DAC Firmware driver registers structures .....	160



15.1.1	DAC_HandleTypeDef .....	160
15.1.2	DAC_ChannelConfTypeDef .....	160
<b>15.2</b>	<b>DAC Firmware driver API description .....</b>	<b>160</b>
15.2.1	DAC Peripheral features .....	160
15.2.2	How to use this driver .....	161
15.2.3	Initialization and de-initialization functions .....	163
15.2.4	IO operation functions .....	163
15.2.5	Peripheral Control functions .....	163
15.2.6	Peripheral State and Errors functions .....	163
15.2.7	Detailed description of functions .....	164
<b>15.3</b>	<b>DAC Firmware driver defines .....</b>	<b>169</b>
15.3.1	DAC .....	169
<b>16</b>	<b>HAL DAC Extension Driver .....</b>	<b>173</b>
16.1	DACEx Firmware driver API description .....	173
16.1.1	How to use this driver .....	173
16.1.2	Extended features functions .....	173
16.1.3	Detailed description of functions .....	173
16.2	DACEx Firmware driver defines .....	177
16.2.1	DACEx .....	177
<b>17</b>	<b>HAL DCMI Generic Driver .....</b>	<b>179</b>
17.1	DCMI Firmware driver registers structures .....	179
17.1.1	DCMI_SyncUnmaskTypeDef .....	179
17.1.2	__DCMI_HandleTypeDef .....	179
17.2	DCMI Firmware driver API description .....	180
17.2.1	How to use this driver .....	180
17.2.2	Initialization and Configuration functions .....	180
17.2.3	IO operation functions .....	181
17.2.4	Peripheral Control functions .....	181
17.2.5	Peripheral State and Errors functions .....	181
17.2.6	Detailed description of functions .....	181
17.3	DCMI Firmware driver defines .....	187
17.3.1	DCMI .....	187
<b>18</b>	<b>HAL DCMI Extension Driver .....</b>	<b>193</b>
18.1	DCMIEx Firmware driver registers structures .....	193
18.1.1	DCMI_CodesInitTypeDef .....	193
18.1.2	DCMI_InitTypeDef .....	193
18.2	DCMIEx Firmware driver defines .....	194

18.2.1	DCMIEx .....	194
<b>19</b>	<b>HAL DFSDM Generic Driver .....</b>	<b>196</b>
19.1	DFSDM Firmware driver registers structures .....	196
19.1.1	DFSDM_Channel_OutputClockTypeDef .....	196
19.1.2	DFSDM_Channel_InputTypeDef .....	196
19.1.3	DFSDM_Channel_SerialInterfaceTypeDef .....	196
19.1.4	DFSDM_Channel_AwdTypeDef .....	197
19.1.5	DFSDM_Channel_InitTypeDef .....	197
19.1.6	DFSDM_Channel_HandleTypeDef .....	197
19.1.7	DFSDM_Filter_RegularParamTypeDef .....	198
19.1.8	DFSDM_Filter_InjectedParamTypeDef .....	198
19.1.9	DFSDM_Filter_FilterParamTypeDef .....	198
19.1.10	DFSDM_Filter_InitTypeDef .....	199
19.1.11	DFSDM_Filter_HandleTypeDef .....	199
19.1.12	DFSDM_Filter_AwdParamTypeDef .....	200
19.1.13	DFSDM_MultiChannelConfigTypeDef .....	200
19.2	DFSDM Firmware driver API description .....	201
19.2.1	How to use this driver .....	201
19.2.2	Channel initialization and de-initialization functions .....	204
19.2.3	Channel operation functions .....	204
19.2.4	Channel state function .....	205
19.2.5	Filter initialization and de-initialization functions .....	205
19.2.6	Filter control functions .....	205
19.2.7	Filter operation functions .....	205
19.2.8	Filter state functions .....	206
19.2.9	Filter MultiChannel operation functions .....	207
19.2.10	Detailed description of functions .....	207
19.3	DFSDM Firmware driver defines .....	228
19.3.1	DFSDM .....	228
<b>20</b>	<b>HAL DMA2D Generic Driver .....</b>	<b>235</b>
20.1	DMA2D Firmware driver registers structures .....	235
20.1.1	DMA2D_CLUTCfgTypeDef .....	235
20.1.2	DMA2D_InitTypeDef .....	235
20.1.3	DMA2D_LayerCfgTypeDef .....	235
20.1.4	__DMA2D_HandleTypeDef .....	236
20.2	DMA2D Firmware driver API description .....	236
20.2.1	How to use this driver .....	236
20.2.2	Initialization and Configuration functions .....	239

20.2.3	IO operation functions . . . . .	239
20.2.4	Peripheral Control functions . . . . .	240
20.2.5	Peripheral State and Errors functions . . . . .	240
20.2.6	Detailed description of functions . . . . .	240
20.3	DMA2D Firmware driver defines . . . . .	250
20.3.1	DMA2D . . . . .	250
<b>21</b>	<b>HAL DMA Generic Driver . . . . .</b>	<b>257</b>
21.1	DMA Firmware driver registers structures . . . . .	257
21.1.1	DMA_InitTypeDef . . . . .	257
21.1.2	__DMA_HandleTypeDef . . . . .	258
21.2	DMA Firmware driver API description. . . . .	259
21.2.1	How to use this driver . . . . .	259
21.2.2	Initialization and de-initialization functions . . . . .	260
21.2.3	IO operation functions . . . . .	260
21.2.4	State and Errors functions. . . . .	260
21.2.5	Detailed description of functions . . . . .	260
21.3	DMA Firmware driver defines . . . . .	264
21.3.1	DMA . . . . .	264
<b>22</b>	<b>HAL DMA Extension Driver . . . . .</b>	<b>269</b>
22.1	DMAEx Firmware driver API description . . . . .	269
22.1.1	How to use this driver . . . . .	269
22.1.2	Extended features functions . . . . .	269
22.1.3	Detailed description of functions . . . . .	269
<b>23</b>	<b>HAL DSI Generic Driver . . . . .</b>	<b>271</b>
23.1	DSI Firmware driver registers structures . . . . .	271
23.1.1	DSI_InitTypeDef . . . . .	271
23.1.2	DSI_PLLInitTypeDef . . . . .	271
23.1.3	DSI_VidCfgTypeDef . . . . .	271
23.1.4	DSI_CmdCfgTypeDef . . . . .	273
23.1.5	DSI_LPcmdTypeDef . . . . .	274
23.1.6	DSI_PHY_TimerTypeDef . . . . .	275
23.1.7	DSI_HOST_TimeoutTypeDef . . . . .	275
23.1.8	DSI_HandleTypeDef . . . . .	276
23.2	DSI Firmware driver API description. . . . .	276
23.2.1	How to use this driver . . . . .	276
23.2.2	Initialization and Configuration functions . . . . .	278
23.2.3	IO operation functions . . . . .	278

23.2.4	Peripheral Control functions . . . . .	278
23.2.5	Peripheral State and Errors functions . . . . .	280
23.2.6	Detailed description of functions . . . . .	280
23.3	DSI Firmware driver defines . . . . .	292
23.3.1	DSI . . . . .	292
<b>24</b>	<b>HAL ETH Generic Driver . . . . .</b>	<b>304</b>
24.1	ETH Firmware driver registers structures . . . . .	304
24.1.1	ETH_InitTypeDef . . . . .	304
24.1.2	ETH_MACInitTypeDef . . . . .	304
24.1.3	ETH_DMAInitTypeDef . . . . .	307
24.1.4	ETH_DMADescTypeDef . . . . .	308
24.1.5	ETH_DMARxFramelInfos . . . . .	309
24.1.6	ETH_HandleTypeDef . . . . .	309
24.2	ETH Firmware driver API description . . . . .	310
24.2.1	How to use this driver . . . . .	310
24.2.2	Initialization and de-initialization functions . . . . .	310
24.2.3	IO operation functions . . . . .	310
24.2.4	Peripheral Control functions . . . . .	311
24.2.5	Peripheral State functions . . . . .	311
24.2.6	Detailed description of functions . . . . .	311
24.3	ETH Firmware driver defines . . . . .	317
24.3.1	ETH . . . . .	317
<b>25</b>	<b>HAL EXTI Generic Driver . . . . .</b>	<b>346</b>
25.1	EXTI Firmware driver registers structures . . . . .	346
25.1.1	EXTI_HandleTypeDef . . . . .	346
25.1.2	EXTI_ConfigTypeDef . . . . .	346
25.2	EXTI Firmware driver API description . . . . .	346
25.2.1	EXTI Peripheral features . . . . .	346
25.2.2	How to use this driver . . . . .	347
25.2.3	Configuration functions . . . . .	347
25.2.4	Detailed description of functions . . . . .	347
25.3	EXTI Firmware driver defines . . . . .	350
25.3.1	EXTI . . . . .	350
<b>26</b>	<b>HAL FLASH Generic Driver . . . . .</b>	<b>353</b>
26.1	FLASH Firmware driver registers structures . . . . .	353
26.1.1	FLASH_ProcessTypeDef . . . . .	353
26.2	FLASH Firmware driver API description . . . . .	353

26.2.1	FLASH peripheral features	353
26.2.2	How to use this driver	353
26.2.3	Programming operation functions	354
26.2.4	Peripheral Control functions	354
26.2.5	Peripheral Errors functions	354
26.2.6	Detailed description of functions	354
26.3	FLASH Firmware driver defines	357
26.3.1	FLASH	357
<b>27</b>	<b>HAL FLASH Extension Driver</b>	<b>363</b>
27.1	FLASHEx Firmware driver registers structures	363
27.1.1	FLASH_EraseInitTypeDef	363
27.1.2	FLASH_OBProgramInitTypeDef	363
27.1.3	FLASH_AdvOBProgramInitTypeDef	364
27.2	FLASHEx Firmware driver API description	364
27.2.1	Flash Extension features	364
27.2.2	How to use this driver	364
27.2.3	Extended programming operation functions	365
27.2.4	Detailed description of functions	365
27.3	FLASHEx Firmware driver defines	368
27.3.1	FLASHEx	368
<b>28</b>	<b>HAL FLASH__RAMFUNC Generic Driver</b>	<b>376</b>
28.1	FLASH__RAMFUNC Firmware driver API description	376
28.1.1	APIs executed from Internal RAM	376
28.1.2	ramfunc functions	376
28.1.3	Detailed description of functions	376
<b>29</b>	<b>HAL FMPI2C Generic Driver</b>	<b>378</b>
29.1	FMPI2C Firmware driver registers structures	378
29.1.1	FMPI2C_InitTypeDef	378
29.1.2	__FMPI2C_HandleTypeDef	378
29.2	FMPI2C Firmware driver API description	379
29.2.1	How to use this driver	379
29.2.2	Initialization and de-initialization functions	384
29.2.3	IO operation functions	385
29.2.4	Peripheral State, Mode and Error functions	386
29.2.5	Detailed description of functions	387
29.3	FMPI2C Firmware driver defines	402
29.3.1	FMPI2C	403

<b>30</b>	<b>HAL FMPI2C Extension Driver</b>	<b>409</b>
30.1	FMPI2CEx Firmware driver API description	409
30.1.1	FMPI2C peripheral Extended features	409
30.1.2	How to use this driver	409
30.1.3	Extended features functions	409
30.1.4	Detailed description of functions	409
30.2	FMPI2CEx Firmware driver defines	411
30.2.1	FMPI2CEx	411
<b>31</b>	<b>HAL GPIO Generic Driver</b>	<b>412</b>
31.1	GPIO Firmware driver registers structures	412
31.1.1	GPIO_InitTypeDef	412
31.2	GPIO Firmware driver API description	412
31.2.1	GPIO Peripheral features	412
31.2.2	How to use this driver	412
31.2.3	Initialization and de-initialization functions	413
31.2.4	IO operation functions	413
31.2.5	Detailed description of functions	413
31.3	GPIO Firmware driver defines	416
31.3.1	GPIO	416
<b>32</b>	<b>HAL GPIO Extension Driver</b>	<b>422</b>
32.1	GPIOEx Firmware driver defines	422
32.1.1	GPIOEx	422
<b>33</b>	<b>HAL HASH Generic Driver</b>	<b>423</b>
33.1	HASH Firmware driver registers structures	423
33.1.1	HASH_InitTypeDef	423
33.1.2	HASH_HandleTypeDef	423
33.2	HASH Firmware driver API description	424
33.2.1	How to use this driver	424
33.2.2	Initialization and de-initialization functions	427
33.2.3	Polling mode HASH processing functions	428
33.2.4	Interrupt mode HASH processing functions	428
33.2.5	DMA mode HASH processing functions	429
33.2.6	Polling mode HMAC processing functions	429
33.2.7	Interrupt mode HMAC processing functions	429
33.2.8	DMA mode HMAC processing functions	429
33.2.9	Peripheral State methods	430
33.2.10	Detailed description of functions	430

33.3	HASH Firmware driver defines .....	448
33.3.1	HASH .....	448
<b>34</b>	<b>HAL HASH Extension Driver .....</b>	<b>453</b>
34.1	HASHEX Firmware driver API description .....	453
34.1.1	HASH peripheral extended features .....	453
34.1.2	Polling mode HASH extended processing functions .....	453
34.1.3	Interruption mode HASH extended processing functions .....	454
34.1.4	DMA mode HASH extended processing functions .....	454
34.1.5	Polling mode HMAC extended processing functions .....	454
34.1.6	Interrupt mode HMAC extended processing functions .....	455
34.1.7	DMA mode HMAC extended processing functions .....	455
34.1.8	Multi-buffer DMA mode HMAC extended processing functions .....	455
34.1.9	Detailed description of functions .....	456
<b>35</b>	<b>HAL HCD Generic Driver .....</b>	<b>473</b>
35.1	HCD Firmware driver registers structures .....	473
35.1.1	HCD_HandleTypeDef .....	473
35.2	HCD Firmware driver API description .....	473
35.2.1	How to use this driver .....	473
35.2.2	Initialization and de-initialization functions .....	474
35.2.3	IO operation functions .....	474
35.2.4	Peripheral Control functions .....	474
35.2.5	Peripheral State functions .....	474
35.2.6	Detailed description of functions .....	474
35.3	HCD Firmware driver defines .....	480
35.3.1	HCD .....	481
<b>36</b>	<b>HAL I2C Generic Driver .....</b>	<b>482</b>
36.1	I2C Firmware driver registers structures .....	482
36.1.1	I2C_InitTypeDef .....	482
36.1.2	__I2C_HandleTypeDef .....	482
36.2	I2C Firmware driver API description .....	483
36.2.1	How to use this driver .....	483
36.2.2	Initialization and de-initialization functions .....	488
36.2.3	IO operation functions .....	488
36.2.4	Peripheral State, Mode and Error functions .....	490
36.2.5	Detailed description of functions .....	490
36.3	I2C Firmware driver defines .....	506
36.3.1	I2C .....	506

<b>37</b>	<b>HAL I2C Extension Driver</b> .....	<b>513</b>
37.1	I2CEx Firmware driver API description .....	513
37.1.1	I2C peripheral extension features .....	513
37.1.2	How to use this driver .....	513
37.1.3	Extension features functions .....	513
37.1.4	Detailed description of functions .....	513
37.2	I2CEx Firmware driver defines .....	514
37.2.1	I2CEx .....	514
<b>38</b>	<b>HAL I2S Generic Driver</b> .....	<b>515</b>
38.1	I2S Firmware driver registers structures .....	515
38.1.1	I2S_InitTypeDef .....	515
38.1.2	__I2S_HandleTypeDef .....	515
38.2	I2S Firmware driver API description .....	516
38.2.1	How to use this driver .....	516
38.2.2	Initialization and de-initialization functions .....	519
38.2.3	IO operation functions .....	519
38.2.4	Peripheral State and Errors functions .....	520
38.2.5	Detailed description of functions .....	520
38.3	I2S Firmware driver defines .....	526
38.3.1	I2S .....	526
<b>39</b>	<b>HAL I2S Extension Driver</b> .....	<b>532</b>
39.1	I2SEx Firmware driver API description .....	532
39.1.1	I2S Extension features .....	532
39.1.2	How to use this driver .....	532
39.1.3	IO operation functions .....	532
39.1.4	Detailed description of functions .....	533
39.2	I2SEx Firmware driver defines .....	535
39.2.1	I2SEx .....	535
<b>40</b>	<b>HAL IRDA Generic Driver</b> .....	<b>538</b>
40.1	IRDA Firmware driver registers structures .....	538
40.1.1	IRDA_InitTypeDef .....	538
40.1.2	IRDA_HandleTypeDef .....	538
40.2	IRDA Firmware driver API description .....	539
40.2.1	How to use this driver .....	539
40.2.2	Callback registration .....	541
40.2.3	Initialization and Configuration functions .....	542
40.2.4	IO operation functions .....	542



	40.2.5	Peripheral State and Errors functions .....	545
	40.2.6	Detailed description of functions .....	545
<b>40.3</b>		<b>IRDA Firmware driver defines .....</b>	<b>554</b>
	40.3.1	IRDA .....	554
<b>41</b>		<b>HAL IWDG Generic Driver .....</b>	<b>561</b>
41.1		IWDG Firmware driver registers structures .....	561
	41.1.1	IWDG_InitTypeDef .....	561
	41.1.2	IWDG_HandleTypeDef .....	561
41.2		IWDG Firmware driver API description .....	561
	41.2.1	IWDG Generic features .....	561
	41.2.2	How to use this driver .....	562
	41.2.3	Initialization and Start functions .....	562
	41.2.4	IO operation functions .....	562
	41.2.5	Detailed description of functions .....	562
41.3		IWDG Firmware driver defines .....	563
	41.3.1	IWDG .....	563
<b>42</b>		<b>HAL LPTIM Generic Driver .....</b>	<b>565</b>
42.1		LPTIM Firmware driver registers structures .....	565
	42.1.1	LPTIM_ClockConfigTypeDef .....	565
	42.1.2	LPTIM_ULPClockConfigTypeDef .....	565
	42.1.3	LPTIM_TriggerConfigTypeDef .....	565
	42.1.4	LPTIM_InitTypeDef .....	565
	42.1.5	LPTIM_HandleTypeDef .....	566
42.2		LPTIM Firmware driver API description .....	566
	42.2.1	How to use this driver .....	566
	42.2.2	Initialization and de-initialization functions .....	568
	42.2.3	LPTIM Start Stop operation functions .....	568
	42.2.4	LPTIM Read operation functions .....	569
	42.2.5	Peripheral State functions .....	569
	42.2.6	Detailed description of functions .....	569
42.3		LPTIM Firmware driver defines .....	580
	42.3.1	LPTIM .....	580
<b>43</b>		<b>HAL LTDC Generic Driver .....</b>	<b>589</b>
43.1		LTDC Firmware driver registers structures .....	589
	43.1.1	LTDC_ColorTypeDef .....	589
	43.1.2	LTDC_InitTypeDef .....	589
	43.1.3	LTDC_LayerCfgTypeDef .....	590

43.1.4	LTDC_HandleTypeDef .....	591
<b>43.2</b>	<b>LTDC Firmware driver API description .....</b>	<b>592</b>
43.2.1	How to use this driver .....	592
43.2.2	Initialization and Configuration functions .....	593
43.2.3	IO operation functions .....	593
43.2.4	Peripheral Control functions .....	594
43.2.5	Peripheral State and Errors functions .....	594
43.2.6	Detailed description of functions .....	595
<b>43.3</b>	<b>LTDC Firmware driver defines .....</b>	<b>606</b>
43.3.1	LTDC .....	606
<b>44</b>	<b>HAL LTDC Extension Driver .....</b>	<b>613</b>
44.1	LTDCEx Firmware driver API description .....	613
44.1.1	Initialization and Configuration functions .....	613
44.1.2	Detailed description of functions .....	613
<b>45</b>	<b>HAL MMC Generic Driver .....</b>	<b>614</b>
45.1	MMC Firmware driver registers structures .....	614
45.1.1	HAL_MMC_CardInfoTypeDef .....	614
45.1.2	MMC_HandleTypeDef .....	614
45.1.3	HAL_MMC_CardCSDTypeDef .....	615
45.1.4	HAL_MMC_CardCIDTypeDef .....	618
45.2	MMC Firmware driver API description .....	618
45.2.1	How to use this driver .....	618
45.2.2	Initialization and de-initialization functions .....	621
45.2.3	IO operation functions .....	621
45.2.4	Peripheral Control functions .....	622
45.2.5	Detailed description of functions .....	622
45.3	MMC Firmware driver defines .....	630
45.3.1	MMC .....	630
<b>46</b>	<b>HAL NAND Generic Driver .....</b>	<b>640</b>
46.1	NAND Firmware driver registers structures .....	640
46.1.1	NAND_IDTypeDef .....	640
46.1.2	NAND_AddressTypeDef .....	640
46.1.3	NAND_DeviceConfigTypeDef .....	640
46.1.4	NAND_HandleTypeDef .....	641
46.2	NAND Firmware driver API description .....	641
46.2.1	How to use this driver .....	641
46.2.2	NAND Initialization and de-initialization functions .....	642

46.2.3	NAND Input and Output functions . . . . .	642
46.2.4	NAND Control functions . . . . .	643
46.2.5	NAND State functions . . . . .	643
46.2.6	Detailed description of functions . . . . .	643
46.3	NAND Firmware driver defines . . . . .	650
46.3.1	NAND . . . . .	650
<b>47</b>	<b>HAL NOR Generic Driver . . . . .</b>	<b>651</b>
47.1	NOR Firmware driver registers structures . . . . .	651
47.1.1	NOR_IDTypeDef . . . . .	651
47.1.2	NOR_CFTypeDef . . . . .	651
47.1.3	NOR_HandleTypeDef . . . . .	651
47.2	NOR Firmware driver API description . . . . .	652
47.2.1	How to use this driver . . . . .	652
47.2.2	NOR Initialization and de_initialization functions . . . . .	653
47.2.3	NOR Input and Output functions . . . . .	653
47.2.4	NOR Control functions . . . . .	653
47.2.5	NOR State functions . . . . .	653
47.2.6	Detailed description of functions . . . . .	654
47.3	NOR Firmware driver defines . . . . .	659
47.3.1	NOR . . . . .	659
<b>48</b>	<b>HAL PCCARD Generic Driver . . . . .</b>	<b>660</b>
48.1	PCCARD Firmware driver registers structures . . . . .	660
48.1.1	PCCARD_HandleTypeDef . . . . .	660
48.2	PCCARD Firmware driver API description . . . . .	660
48.2.1	How to use this driver . . . . .	660
48.2.2	PCCARD Initialization and de-initialization functions . . . . .	661
48.2.3	PCCARD Input and Output functions . . . . .	661
48.2.4	PCCARD State functions . . . . .	661
48.2.5	Detailed description of functions . . . . .	662
48.3	PCCARD Firmware driver defines . . . . .	666
48.3.1	PCCARD . . . . .	666
<b>49</b>	<b>HAL PCD Generic Driver . . . . .</b>	<b>667</b>
49.1	PCD Firmware driver registers structures . . . . .	667
49.1.1	PCD_HandleTypeDef . . . . .	667
49.2	PCD Firmware driver API description . . . . .	668
49.2.1	How to use this driver . . . . .	668
49.2.2	Initialization and de-initialization functions . . . . .	668

49.2.3	IO operation functions . . . . .	668
49.2.4	Peripheral Control functions . . . . .	669
49.2.5	Peripheral State functions . . . . .	669
49.2.6	Detailed description of functions . . . . .	669
49.3	PCD Firmware driver defines . . . . .	677
49.3.1	PCD . . . . .	677
<b>50</b>	<b>HAL PCD Extension Driver . . . . .</b>	<b>679</b>
50.1	PCDEx Firmware driver API description . . . . .	679
50.1.1	Extended features functions . . . . .	679
50.1.2	Detailed description of functions . . . . .	679
<b>51</b>	<b>HAL PWR Generic Driver . . . . .</b>	<b>681</b>
51.1	PWR Firmware driver registers structures . . . . .	681
51.1.1	PWR_PVDTypeDef . . . . .	681
51.2	PWR Firmware driver API description . . . . .	681
51.2.1	Initialization and de-initialization functions . . . . .	681
51.2.2	Peripheral Control functions . . . . .	681
51.2.3	Detailed description of functions . . . . .	683
51.3	PWR Firmware driver defines . . . . .	688
51.3.1	PWR . . . . .	688
<b>52</b>	<b>HAL PWR Extension Driver . . . . .</b>	<b>693</b>
52.1	PWREx Firmware driver API description . . . . .	693
52.1.1	Peripheral extended features functions . . . . .	693
52.1.2	Detailed description of functions . . . . .	694
52.2	PWREx Firmware driver defines . . . . .	697
52.2.1	PWREx . . . . .	697
<b>53</b>	<b>HAL QSPI Generic Driver . . . . .</b>	<b>700</b>
53.1	QSPI Firmware driver registers structures . . . . .	700
53.1.1	QSPI_InitTypeDef . . . . .	700
53.1.2	QSPI_HandleTypeDef . . . . .	700
53.1.3	QSPI_CommandTypeDef . . . . .	701
53.1.4	QSPI_AutoPollingTypeDef . . . . .	701
53.1.5	QSPI_MemoryMappedTypeDef . . . . .	702
53.2	QSPI Firmware driver API description . . . . .	702
53.2.1	How to use this driver . . . . .	702
53.2.2	Initialization and Configuration functions . . . . .	705
53.2.3	IO operation functions . . . . .	705
53.2.4	Peripheral Control and State functions . . . . .	706

	53.2.5	Detailed description of functions .....	706
53.3		QSPI Firmware driver defines .....	716
	53.3.1	QSPI .....	716
<b>54</b>		<b>HAL RCC Generic Driver .....</b>	<b>723</b>
54.1		RCC Firmware driver registers structures .....	723
	54.1.1	RCC_OscInitTypeDef .....	723
	54.1.2	RCC_ClkInitTypeDef .....	723
54.2		RCC Firmware driver API description .....	724
	54.2.1	RCC specific features .....	724
	54.2.2	RCC Limitations .....	724
	54.2.3	Initialization and de-initialization functions .....	724
	54.2.4	Peripheral Control functions .....	725
	54.2.5	Detailed description of functions .....	726
54.3		RCC Firmware driver defines .....	730
	54.3.1	RCC .....	730
<b>55</b>		<b>HAL RCC Extension Driver .....</b>	<b>750</b>
55.1		RCCEX Firmware driver registers structures .....	750
	55.1.1	RCC_PLLInitTypeDef .....	750
	55.1.2	RCC_PLLI2SInitTypeDef .....	750
	55.1.3	RCC_PLLSAIInitTypeDef .....	751
	55.1.4	RCC_PeriphCLKInitTypeDef .....	751
55.2		RCCEX Firmware driver API description .....	752
	55.2.1	Extended Peripheral Control functions .....	752
	55.2.2	Detailed description of functions .....	752
55.3		RCCEX Firmware driver defines .....	754
	55.3.1	RCCEX .....	755
<b>56</b>		<b>HAL RNG Generic Driver .....</b>	<b>779</b>
56.1		RNG Firmware driver registers structures .....	779
	56.1.1	RNG_HandleTypeDef .....	779
56.2		RNG Firmware driver API description .....	779
	56.2.1	How to use this driver .....	779
	56.2.2	Callback registration .....	779
	56.2.3	Initialization and configuration functions .....	780
	56.2.4	Peripheral Control functions .....	780
	56.2.5	Peripheral State functions .....	780
	56.2.6	Detailed description of functions .....	781
56.3		RNG Firmware driver defines .....	784

56.3.1	RNG .....	784
<b>57</b>	<b>HAL RTC Generic Driver .....</b>	<b>788</b>
57.1	RTC Firmware driver registers structures .....	788
57.1.1	RTC_InitTypeDef .....	788
57.1.2	RTC_TimeTypeDef .....	788
57.1.3	RTC_DateTypeDef .....	789
57.1.4	RTC_AlarmTypeDef .....	789
57.1.5	RTC_HandleTypeDef .....	790
57.2	RTC Firmware driver API description .....	790
57.2.1	Backup Domain Operating Condition .....	790
57.2.2	Backup Domain Reset .....	791
57.2.3	Backup Domain Access .....	791
57.2.4	How to use this driver .....	791
57.2.5	RTC and low power modes .....	791
57.2.6	Initialization and de-initialization functions .....	792
57.2.7	RTC Time and Date functions .....	793
57.2.8	RTC Alarm functions .....	793
57.2.9	Peripheral Control functions .....	793
57.2.10	Peripheral State functions .....	793
57.2.11	Detailed description of functions .....	793
57.3	RTC Firmware driver defines .....	800
57.3.1	RTC .....	800
<b>58</b>	<b>HAL RTC Extension Driver .....</b>	<b>810</b>
58.1	RTCEX Firmware driver registers structures .....	810
58.1.1	RTC_TamperTypeDef .....	810
58.2	RTCEX Firmware driver API description .....	810
58.2.1	How to use this driver .....	810
58.2.2	RTC TimeStamp and Tamper functions .....	811
58.2.3	RTC Wake-up functions .....	811
58.2.4	Extension Peripheral Control functions .....	812
58.2.5	Extended features functions .....	812
58.2.6	Detailed description of functions .....	812
58.3	RTCEX Firmware driver defines .....	823
58.3.1	RTCEX .....	823
<b>59</b>	<b>HAL SAI Generic Driver .....</b>	<b>838</b>
59.1	SAI Firmware driver registers structures .....	838
59.1.1	SAI_InitTypeDef .....	838

59.1.2	SAI_FrameInitTypeDef	839
59.1.3	SAI_SlotInitTypeDef	840
59.1.4	__SAI_HandleTypeDef	840
<b>59.2</b>	<b>SAI Firmware driver API description</b>	<b>841</b>
59.2.1	How to use this driver	841
59.2.2	Initialization and de-initialization functions	843
59.2.3	IO operation functions	844
59.2.4	Peripheral State and Errors functions	845
59.2.5	Detailed description of functions	845
<b>59.3</b>	<b>SAI Firmware driver defines</b>	<b>852</b>
59.3.1	SAI	852
<b>60</b>	<b>HAL SAI Extension Driver</b>	<b>860</b>
<b>60.1</b>	<b>SAIEx Firmware driver API description</b>	<b>860</b>
60.1.1	SAI peripheral extension features	860
60.1.2	How to use this driver	860
60.1.3	Extension features Functions	860
60.1.4	Detailed description of functions	860
<b>60.2</b>	<b>SAIEx Firmware driver defines</b>	<b>860</b>
60.2.1	SAIEx	861
<b>61</b>	<b>HAL SD Generic Driver</b>	<b>862</b>
<b>61.1</b>	<b>SD Firmware driver registers structures</b>	<b>862</b>
61.1.1	HAL_SD_CardInfoTypeDef	862
61.1.2	SD_HandleTypeDef	862
61.1.3	HAL_SD_CardCSDTypeDef	863
61.1.4	HAL_SD_CardCIDTypeDef	866
61.1.5	HAL_SD_CardStatusTypeDef	866
<b>61.2</b>	<b>SD Firmware driver API description</b>	<b>867</b>
61.2.1	How to use this driver	867
61.2.2	Initialization and de-initialization functions	870
61.2.3	IO operation functions	870
61.2.4	Peripheral Control functions	870
61.2.5	Detailed description of functions	871
<b>61.3</b>	<b>SD Firmware driver defines</b>	<b>879</b>
61.3.1	SD	879
<b>62</b>	<b>HAL SDRAM Generic Driver</b>	<b>889</b>
<b>62.1</b>	<b>SDRAM Firmware driver registers structures</b>	<b>889</b>
62.1.1	SDRAM_HandleTypeDef	889

62.2	SDRAM Firmware driver API description .....	889
62.2.1	How to use this driver .....	889
62.2.2	SDRAM Initialization and de_initialization functions .....	890
62.2.3	SDRAM Input and Output functions .....	890
62.2.4	SDRAM Control functions .....	891
62.2.5	SDRAM State functions .....	891
62.2.6	Detailed description of functions .....	891
62.3	SDRAM Firmware driver defines .....	898
62.3.1	SDRAM .....	898
<b>63</b>	<b>HAL SMARTCARD Generic Driver .....</b>	<b>899</b>
63.1	SMARTCARD Firmware driver registers structures .....	899
63.1.1	SMARTCARD_InitTypeDef .....	899
63.1.2	__SMARTCARD_HandleTypeDef .....	900
63.2	SMARTCARD Firmware driver API description .....	901
63.2.1	How to use this driver .....	901
63.2.2	Callback registration .....	902
63.2.3	Initialization and Configuration functions .....	903
63.2.4	IO operation functions .....	904
63.2.5	Peripheral State and Errors functions .....	906
63.2.6	Detailed description of functions .....	906
63.3	SMARTCARD Firmware driver defines .....	914
63.3.1	SMARTCARD .....	914
<b>64</b>	<b>HAL SMBUS Generic Driver .....</b>	<b>922</b>
64.1	SMBUS Firmware driver registers structures .....	922
64.1.1	SMBUS_InitTypeDef .....	922
64.1.2	__SMBUS_HandleTypeDef .....	922
64.2	SMBUS Firmware driver API description .....	923
64.2.1	How to use this driver .....	923
64.2.2	Initialization and de-initialization functions .....	926
64.2.3	IO operation functions .....	926
64.2.4	Peripheral State, Mode and Error functions .....	927
64.2.5	Detailed description of functions .....	927
64.3	SMBUS Firmware driver defines .....	936
64.3.1	SMBUS .....	936
<b>65</b>	<b>HAL SPDIFRX Generic Driver .....</b>	<b>942</b>
65.1	SPDIFRX Firmware driver registers structures .....	942
65.1.1	SPDIFRX_InitTypeDef .....	942



65.1.2	SPDIFRX_SetDataFormatTypeDef . . . . .	942
65.1.3	SPDIFRX_HandleTypeDef . . . . .	943
<b>65.2</b>	<b>SPDIFRX Firmware driver API description . . . . .</b>	<b>944</b>
65.2.1	How to use this driver . . . . .	944
65.2.2	Initialization and de-initialization functions . . . . .	945
65.2.3	IO operation functions . . . . .	946
65.2.4	Peripheral State and Errors functions . . . . .	946
65.2.5	Detailed description of functions . . . . .	946
<b>65.3</b>	<b>SPDIFRX Firmware driver defines . . . . .</b>	<b>952</b>
65.3.1	SPDIFRX . . . . .	952
<b>66</b>	<b>HAL SPI Generic Driver . . . . .</b>	<b>957</b>
66.1	SPI Firmware driver registers structures . . . . .	957
66.1.1	SPI_InitTypeDef . . . . .	957
66.1.2	__SPI_HandleTypeDef . . . . .	958
66.2	SPI Firmware driver API description . . . . .	959
66.2.1	How to use this driver . . . . .	959
66.2.2	Initialization and de-initialization functions . . . . .	960
66.2.3	IO operation functions . . . . .	961
66.2.4	Peripheral State and Errors functions . . . . .	962
66.2.5	Detailed description of functions . . . . .	962
66.3	SPI Firmware driver defines . . . . .	970
66.3.1	SPI . . . . .	970
<b>67</b>	<b>HAL SRAM Generic Driver . . . . .</b>	<b>976</b>
67.1	SRAM Firmware driver registers structures . . . . .	976
67.1.1	SRAM_HandleTypeDef . . . . .	976
67.2	SRAM Firmware driver API description . . . . .	976
67.2.1	How to use this driver . . . . .	976
67.2.2	SRAM Initialization and de_initialization functions . . . . .	977
67.2.3	SRAM Input and Output functions . . . . .	977
67.2.4	SRAM Control functions . . . . .	978
67.2.5	SRAM State functions . . . . .	978
67.2.6	Detailed description of functions . . . . .	978
67.3	SRAM Firmware driver defines . . . . .	983
67.3.1	SRAM . . . . .	983
<b>68</b>	<b>HAL TIM Generic Driver . . . . .</b>	<b>984</b>
68.1	TIM Firmware driver registers structures . . . . .	984
68.1.1	TIM_Base_InitTypeDef . . . . .	984

68.1.2	TIM_OC_InitTypeDef .....	984
68.1.3	TIM_OnePulse_InitTypeDef .....	985
68.1.4	TIM_IC_InitTypeDef .....	986
68.1.5	TIM_Encoder_InitTypeDef .....	986
68.1.6	TIM_ClockConfigTypeDef .....	987
68.1.7	TIM_ClearInputConfigTypeDef .....	987
68.1.8	TIM_MasterConfigTypeDef .....	988
68.1.9	TIM_SlaveConfigTypeDef .....	988
68.1.10	TIM_BreakDeadTimeConfigTypeDef .....	988
68.1.11	TIM_HandleTypeDef .....	989
<b>68.2</b>	<b>TIM Firmware driver API description .....</b>	<b>990</b>
68.2.1	TIMER Generic features .....	990
68.2.2	How to use this driver .....	990
68.2.3	Time Base functions .....	992
68.2.4	TIM Output Compare functions .....	992
68.2.5	TIM PWM functions .....	993
68.2.6	TIM Input Capture functions .....	993
68.2.7	TIM One Pulse functions .....	994
68.2.8	TIM Encoder functions .....	994
68.2.9	TIM Callbacks functions .....	994
68.2.10	Detailed description of functions .....	995
<b>68.3</b>	<b>TIM Firmware driver defines .....</b>	<b>1028</b>
68.3.1	TIM .....	1028
<b>69</b>	<b>HAL TIM Extension Driver .....</b>	<b>1051</b>
<b>69.1</b>	<b>TIMEx Firmware driver registers structures .....</b>	<b>1051</b>
69.1.1	TIM_HallSensor_InitTypeDef .....	1051
<b>69.2</b>	<b>TIMEx Firmware driver API description .....</b>	<b>1051</b>
69.2.1	TIMER Extended features .....	1051
69.2.2	How to use this driver .....	1051
69.2.3	Timer Hall Sensor functions .....	1052
69.2.4	Timer Complementary Output Compare functions .....	1052
69.2.5	Timer Complementary PWM functions .....	1053
69.2.6	Timer Complementary One Pulse functions .....	1053
69.2.7	Peripheral Control functions .....	1053
69.2.8	Extended Callbacks functions .....	1054
69.2.9	Extended Peripheral State functions .....	1054
69.2.10	Detailed description of functions .....	1054
<b>69.3</b>	<b>TIMEx Firmware driver defines .....</b>	<b>1067</b>

69.3.1	TIMEx .....	1067
<b>70</b>	<b>HAL UART Generic Driver .....</b>	<b>1068</b>
70.1	UART Firmware driver registers structures .....	1068
70.1.1	UART_InitTypeDef .....	1068
70.1.2	__UART_HandleTypeDef .....	1068
70.2	UART Firmware driver API description .....	1069
70.2.1	How to use this driver .....	1069
70.2.2	Callback registration .....	1070
70.2.3	Initialization and Configuration functions .....	1073
70.2.4	IO operation functions .....	1073
70.2.5	Peripheral Control functions .....	1074
70.2.6	Peripheral State and Errors functions .....	1074
70.2.7	Detailed description of functions .....	1074
70.3	UART Firmware driver defines .....	1089
70.3.1	UART .....	1089
<b>71</b>	<b>HAL USART Generic Driver .....</b>	<b>1098</b>
71.1	USART Firmware driver registers structures .....	1098
71.1.1	USART_InitTypeDef .....	1098
71.1.2	__USART_HandleTypeDef .....	1098
71.2	USART Firmware driver API description .....	1099
71.2.1	How to use this driver .....	1099
71.2.2	Callback registration .....	1101
71.2.3	Initialization and Configuration functions .....	1102
71.2.4	IO operation functions .....	1102
71.2.5	Peripheral State and Errors functions .....	1104
71.2.6	Detailed description of functions .....	1104
71.3	USART Firmware driver defines .....	1113
71.3.1	USART .....	1113
<b>72</b>	<b>HAL WWDG Generic Driver .....</b>	<b>1119</b>
72.1	WWDG Firmware driver registers structures .....	1119
72.1.1	WWDG_InitTypeDef .....	1119
72.1.2	WWDG_HandleTypeDef .....	1119
72.2	WWDG Firmware driver API description .....	1119
72.2.1	Initialization and Configuration functions .....	1119
72.2.2	IO operation functions .....	1119
72.2.3	Detailed description of functions .....	1120
72.3	WWDG Firmware driver defines .....	1121

	72.3.1	WWDG .....	1121
<b>73</b>	<b>LL ADC Generic Driver.....</b>		<b>1124</b>
73.1	ADC Firmware driver registers structures .....		1124
	73.1.1	LL_ADC_CommonInitTypeDef .....	1124
	73.1.2	LL_ADC_InitTypeDef .....	1124
	73.1.3	LL_ADC_REG_InitTypeDef.....	1124
	73.1.4	LL_ADC_INJ_InitTypeDef.....	1125
73.2	ADC Firmware driver API description .....		1126
	73.2.1	Detailed description of functions .....	1126
73.3	ADC Firmware driver defines.....		1192
	73.3.1	ADC .....	1192
<b>74</b>	<b>LL BUS Generic Driver.....</b>		<b>1224</b>
74.1	BUS Firmware driver API description .....		1224
	74.1.1	Detailed description of functions .....	1224
74.2	BUS Firmware driver defines.....		1270
	74.2.1	BUS .....	1270
<b>75</b>	<b>LL CORTEX Generic Driver .....</b>		<b>1274</b>
75.1	CORTEX Firmware driver API description .....		1274
	75.1.1	Detailed description of functions .....	1274
75.2	CORTEX Firmware driver defines.....		1282
	75.2.1	CORTEX.....	1282
<b>76</b>	<b>LL CRC Generic Driver.....</b>		<b>1286</b>
76.1	CRC Firmware driver API description.....		1286
	76.1.1	Detailed description of functions .....	1286
76.2	CRC Firmware driver defines .....		1288
	76.2.1	CRC .....	1288
<b>77</b>	<b>LL DAC Generic Driver.....</b>		<b>1289</b>
77.1	DAC Firmware driver registers structures .....		1289
	77.1.1	LL_DAC_InitTypeDef .....	1289
77.2	DAC Firmware driver API description.....		1289
	77.2.1	Detailed description of functions .....	1289
77.3	DAC Firmware driver defines.....		1308
	77.3.1	DAC .....	1309
<b>78</b>	<b>LL DMA2D Generic Driver.....</b>		<b>1314</b>
78.1	DMA2D Firmware driver registers structures.....		1314
	78.1.1	LL_DMA2D_InitTypeDef .....	1314

78.1.2	LL_DMA2D_LayerCfgTypeDef .....	1316
78.1.3	LL_DMA2D_ColorTypeDef .....	1317
78.2	DMA2D Firmware driver API description .....	1319
78.2.1	Detailed description of functions .....	1319
78.3	DMA2D Firmware driver defines .....	1358
78.3.1	DMA2D .....	1358
<b>79</b>	<b>LL DMA Generic Driver .....</b>	<b>1361</b>
79.1	DMA Firmware driver registers structures .....	1361
79.1.1	LL_DMA_InitTypeDef .....	1361
79.2	DMA Firmware driver API description .....	1362
79.2.1	Detailed description of functions .....	1363
79.3	DMA Firmware driver defines .....	1424
79.3.1	DMA .....	1424
<b>80</b>	<b>LL FMPI2C Generic Driver .....</b>	<b>1430</b>
80.1	FMPI2C Firmware driver registers structures .....	1430
80.1.1	LL_FMPI2C_InitTypeDef .....	1430
80.2	FMPI2C Firmware driver API description .....	1430
80.2.1	Detailed description of functions .....	1430
80.3	FMPI2C Firmware driver defines .....	1477
80.3.1	FMPI2C .....	1477
<b>81</b>	<b>LL EXTI Generic Driver .....</b>	<b>1483</b>
81.1	EXTI Firmware driver registers structures .....	1483
81.1.1	LL_EXTI_InitTypeDef .....	1483
81.2	EXTI Firmware driver API description .....	1483
81.2.1	Detailed description of functions .....	1483
81.3	EXTI Firmware driver defines .....	1500
81.3.1	EXTI .....	1500
<b>82</b>	<b>LL GPIO Generic Driver .....</b>	<b>1503</b>
82.1	GPIO Firmware driver registers structures .....	1503
82.1.1	LL_GPIO_InitTypeDef .....	1503
82.2	GPIO Firmware driver API description .....	1503
82.2.1	Detailed description of functions .....	1503
82.3	GPIO Firmware driver defines .....	1523
82.3.1	GPIO .....	1523
<b>83</b>	<b>LL I2C Generic Driver .....</b>	<b>1527</b>
83.1	I2C Firmware driver registers structures .....	1527

	83.1.1	LL_I2C_InitTypeDef .....	1527
<b>83.2</b>		I2C Firmware driver API description .....	1527
	83.2.1	Detailed description of functions .....	1528
<b>83.3</b>		I2C Firmware driver defines .....	1566
	83.3.1	I2C .....	1566
<b>84</b>		<b>LL IWDG Generic Driver .....</b>	<b>1572</b>
	84.1	IWDG Firmware driver API description .....	1572
	84.1.1	Detailed description of functions .....	1572
	84.2	IWDG Firmware driver defines .....	1575
	84.2.1	IWDG .....	1575
<b>85</b>		<b>LL LPTIM Generic Driver .....</b>	<b>1577</b>
	85.1	LPTIM Firmware driver registers structures .....	1577
	85.1.1	LL_LPTIM_InitTypeDef .....	1577
	85.2	LPTIM Firmware driver API description .....	1577
	85.2.1	Detailed description of functions .....	1577
	85.3	LPTIM Firmware driver defines .....	1604
	85.3.1	LPTIM .....	1604
<b>86</b>		<b>LL PWR Generic Driver .....</b>	<b>1609</b>
	86.1	PWR Firmware driver API description .....	1609
	86.1.1	Detailed description of functions .....	1609
	86.2	PWR Firmware driver defines .....	1625
	86.2.1	PWR .....	1625
<b>87</b>		<b>LL RCC Generic Driver .....</b>	<b>1628</b>
	87.1	RCC Firmware driver registers structures .....	1628
	87.1.1	LL_RCC_ClocksTypeDef .....	1628
	87.2	RCC Firmware driver API description .....	1628
	87.2.1	Detailed description of functions .....	1628
	87.3	RCC Firmware driver defines .....	1711
	87.3.1	RCC .....	1711
<b>88</b>		<b>LL RNG Generic Driver .....</b>	<b>1765</b>
	88.1	RNG Firmware driver API description .....	1765
	88.1.1	Detailed description of functions .....	1765
	88.2	RNG Firmware driver defines .....	1769
	88.2.1	RNG .....	1769
<b>89</b>		<b>LL RTC Generic Driver .....</b>	<b>1771</b>
	89.1	RTC Firmware driver registers structures .....	1771

89.1.1	LL_RTC_InitTypeDef . . . . .	1771
89.1.2	LL_RTC_TimeTypeDef . . . . .	1771
89.1.3	LL_RTC_DateTypeDef . . . . .	1771
89.1.4	LL_RTC_AlarmTypeDef . . . . .	1772
89.2	RTC Firmware driver API description . . . . .	1772
89.2.1	Detailed description of functions . . . . .	1772
89.3	RTC Firmware driver defines . . . . .	1852
89.3.1	RTC . . . . .	1852
<b>90</b>	<b>LL SPI Generic Driver . . . . .</b>	<b>1863</b>
90.1	SPI Firmware driver registers structures . . . . .	1863
90.1.1	LL_SPI_InitTypeDef . . . . .	1863
90.1.2	LL_I2S_InitTypeDef . . . . .	1864
90.2	SPI Firmware driver API description . . . . .	1864
90.2.1	Detailed description of functions . . . . .	1864
90.3	SPI Firmware driver defines . . . . .	1902
90.3.1	SPI . . . . .	1903
<b>91</b>	<b>LL SYSTEM Generic Driver . . . . .</b>	<b>1906</b>
91.1	SYSTEM Firmware driver API description . . . . .	1906
91.1.1	Detailed description of functions . . . . .	1906
91.2	SYSTEM Firmware driver defines . . . . .	1922
91.2.1	SYSTEM . . . . .	1922
<b>92</b>	<b>LL TIM Generic Driver . . . . .</b>	<b>1927</b>
92.1	TIM Firmware driver registers structures . . . . .	1927
92.1.1	LL_TIM_InitTypeDef . . . . .	1927
92.1.2	LL_TIM_OC_InitTypeDef . . . . .	1927
92.1.3	LL_TIM_IC_InitTypeDef . . . . .	1928
92.1.4	LL_TIM_ENCODER_InitTypeDef . . . . .	1929
92.1.5	LL_TIM_HALLSENSOR_InitTypeDef . . . . .	1929
92.1.6	LL_TIM_BDTR_InitTypeDef . . . . .	1930
92.2	TIM Firmware driver API description . . . . .	1931
92.2.1	Detailed description of functions . . . . .	1931
92.3	TIM Firmware driver defines . . . . .	2006
92.3.1	TIM . . . . .	2006
<b>93</b>	<b>LL USART Generic Driver . . . . .</b>	<b>2020</b>
93.1	USART Firmware driver registers structures . . . . .	2020
93.1.1	LL_USART_InitTypeDef . . . . .	2020
93.1.2	LL_USART_ClockInitTypeDef . . . . .	2020

93.2	USART Firmware driver API description .....	2021
93.2.1	Detailed description of functions .....	2021
93.3	USART Firmware driver defines .....	2072
93.3.1	USART .....	2072
<b>94</b>	<b>LL UTILS Generic Driver .....</b>	<b>2077</b>
94.1	UTILS Firmware driver registers structures .....	2077
94.1.1	LL_UTILS_PLLInitTypeDef .....	2077
94.1.2	LL_UTILS_ClkInitTypeDef .....	2077
94.2	UTILS Firmware driver API description .....	2077
94.2.1	System Configuration functions .....	2077
94.2.2	Detailed description of functions .....	2078
94.3	UTILS Firmware driver defines .....	2081
94.3.1	UTILS .....	2081
<b>95</b>	<b>LL WWDG Generic Driver .....</b>	<b>2083</b>
95.1	WWDG Firmware driver API description .....	2083
95.1.1	Detailed description of functions .....	2083
95.2	WWDG Firmware driver defines .....	2087
95.2.1	WWDG .....	2087
<b>96</b>	<b>FAQs .....</b>	<b>2089</b>
	<b>Revision history .....</b>	<b>2092</b>
	<b>List of tables .....</b>	<b>2121</b>
	<b>List of figures .....</b>	<b>2122</b>



## List of tables

<b>Table 1.</b>	Acronyms and definitions . . . . .	4
<b>Table 2.</b>	HAL driver files . . . . .	8
<b>Table 3.</b>	User-application files . . . . .	8
<b>Table 4.</b>	API classification . . . . .	13
<b>Table 5.</b>	List of devices supported by HAL drivers. . . . .	14
<b>Table 6.</b>	HAL API naming rules . . . . .	16
<b>Table 7.</b>	Macros handling interrupts and specific clock configurations . . . . .	17
<b>Table 8.</b>	Callback functions . . . . .	18
<b>Table 9.</b>	HAL generic APIs . . . . .	19
<b>Table 10.</b>	HAL extension APIs . . . . .	19
<b>Table 11.</b>	Define statements used for HAL configuration . . . . .	24
<b>Table 12.</b>	Description of GPIO_InitTypeDef structure . . . . .	26
<b>Table 13.</b>	Description of EXTI configuration macros . . . . .	28
<b>Table 14.</b>	MSP functions. . . . .	32
<b>Table 15.</b>	Timeout values . . . . .	35
<b>Table 16.</b>	LL driver files. . . . .	39
<b>Table 17.</b>	Common peripheral initialization functions. . . . .	41
<b>Table 18.</b>	Optional peripheral initialization functions . . . . .	41
<b>Table 19.</b>	Specific Interrupt, DMA request and status flags management . . . . .	42
<b>Table 20.</b>	Available function formats . . . . .	42
<b>Table 21.</b>	Peripheral clock activation/deactivation management . . . . .	43
<b>Table 22.</b>	Peripheral activation/deactivation management . . . . .	43
<b>Table 23.</b>	Peripheral configuration management. . . . .	43
<b>Table 24.</b>	Peripheral register management . . . . .	43
<b>Table 25.</b>	Document revision history . . . . .	.2092

## List of figures

<b>Figure 1.</b>	Example of project template . . . . .	10
<b>Figure 2.</b>	Adding device-specific functions . . . . .	20
<b>Figure 3.</b>	Adding family-specific functions . . . . .	20
<b>Figure 4.</b>	Adding new peripherals . . . . .	21
<b>Figure 5.</b>	Updating existing APIs. . . . .	21
<b>Figure 6.</b>	File inclusion model. . . . .	23
<b>Figure 7.</b>	HAL driver model . . . . .	30
<b>Figure 8.</b>	Low-layer driver folders . . . . .	39
<b>Figure 9.</b>	Low-layer driver CMSIS files . . . . .	40

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2021 STMicroelectronics – All rights reserved