UM1725

User manual

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 CodeSonar® static analysis tool. It is fully documented.

It is compliant with MISRA C®: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® Cortex®-M processor.

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

2 Acronyms and definitions

Table 1. Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex microcontroller software interface standard
COMP	Comparator
CORDIC	Trigonometric calculation unit
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
CSS	Clock security system
DAC	Digital to analog converter
DLYB	Delay block
DCMI	Digital camera interface
DFSDM	Digital filter sigma delta modulator
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
DSI	Display serial interface
DTS	Digital temperature sensor
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 controler
LTDC	LCD TFT display controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MDIOS	Management data input/output (MDIO) slave
MDMA	Master direct memory access
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND Flash memory
NOR	NOR Flash memory
NVIC	Nested vectored interrupt controller
OCTOSPI	Octo-SPI interface
OPAMP	Operational amplifier
OTFDEC	On-the-fly decryption engine
OTG-FS	USB on-the-go full-speed
PKA	Public key accelerator
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PSSI	Parallel synchronous slave interface
PWR	Power controller
QSPI	Quad-SPI Flash memory
RAMECC	RAM ECC monitoring
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
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® and Power Delivery interface
USART	Universal synchronous receiver/transmitter
VREFBUF	Voltage reference buffer
WWDG	Window watchdog
USB	Universal serial bus

3 Overview of HAL drivers

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

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

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

The HAL main features are the following:

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

3.1 HAL and user-application files

3.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
stm32f4xx_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32f4xx_hal_adc.c, stm32f4xx_hal_irda.c.*
stm32f4xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32f4xx_hal_adc.h,stm32f4xx_hal_irda.h.*
stm32f4xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32f4xx_hal_adc_ex.c,stm32f4xx_hal_flash_ex.c.*
stm32f4xx_hal_ppp_ex.h	Header file of the extension C file It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f4xx_hal_adc_ex.h,stm32f4xx_hal_flash_ex.h.*
stm32f4xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32f4xx_hal.h	stm32f4xx_hal.c header file
stm32f4xx_hal_msp_template.c	Template file to be copied to the user application folder It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f4xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application
stm32f4xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros

3.1.2 User-application files

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

Table 3. User-application files

File	Description
system_stm32f4xx.c	This file contains SystemInit() that is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM and configuring the FSMC/FMC (when available) to use the external SRAM or SDRAM mounted on the evaluation board as data memory.
startup_stm32f4xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32f4xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly adapting the stack/heap size to fit the application requirements.
stm32f4xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.

File	Description
stm32f4xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32f4xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32f4xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

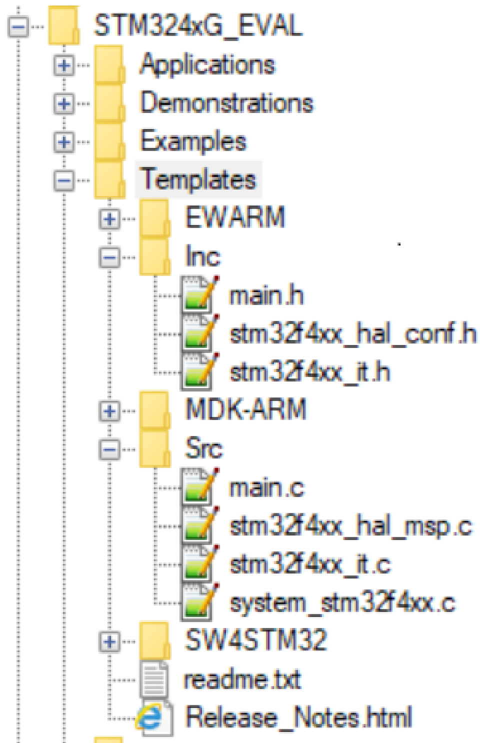
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  – HAL is initialized
  – 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 wether the Receive or Transmit mode is enabled or disabled.*/
uint32_t HwFlowCtl; /*!< Specifies wether the hardware flow control mode is enabled or
disabled.*/
uint32_t OverSampling; /*!< Specifies wether 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
Common APIs	X	X[(1)]
Family specific APIs	-	X
Device specific APIs	-	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_dfsdm.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_fmpi2c.c	No	No	No	No	No	No	No	No	No	No	Yes	No	No	Yes	Yes
stm32f4xx_hal_fmpi2c_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_ltdc.c	No	No	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_ltdc_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	*stm32f4xx_hal_ppp (c/h)*	*stm32f4xx_hal_ppp_ex (c/h)*	*stm32f4xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32F4 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in the stm32f2xxx.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
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the 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
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in run time the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in run time the error that occurred during IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

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

The extension model consists of an additional file, 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
HAL_ADCEx_InjectedStart()	This function starts injected channel ADC conversions when the polling method is used
HAL_ADCEx_InjectedStop()	This function stops injected channel ADC conversions when the polling method is used

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

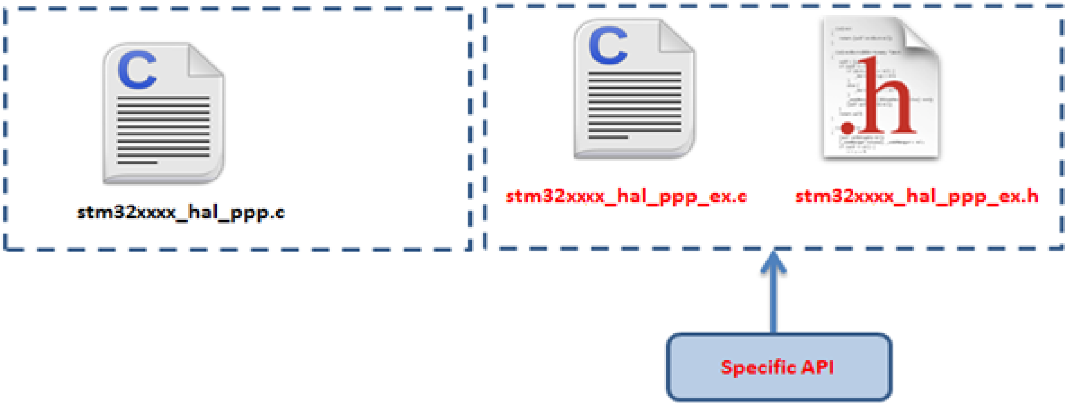
3.7.2 HAL extension model cases

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



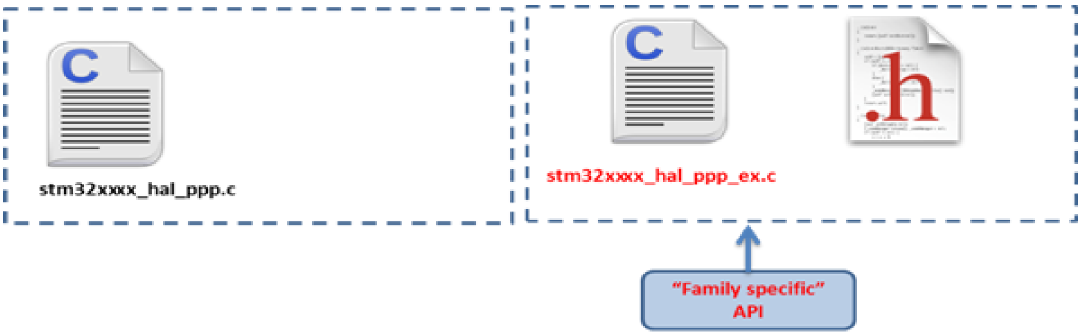
Example: stm32f4xx_hal_flash_ex.c/h

```
#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

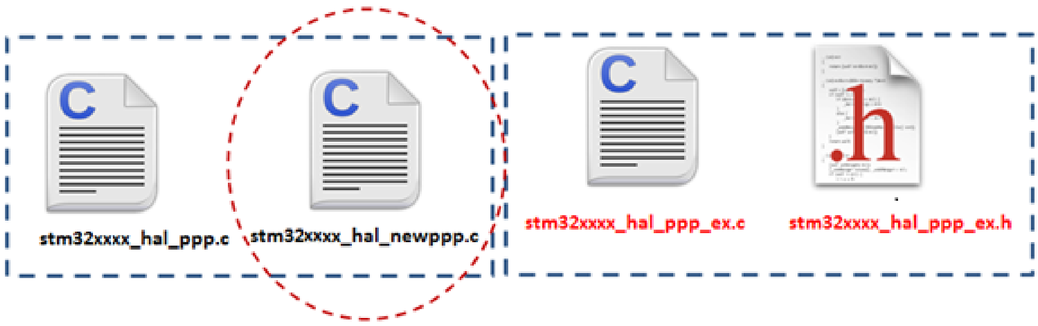
```
HAL_StatusTypeDef HAL_ADCEx_InjectedStop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

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

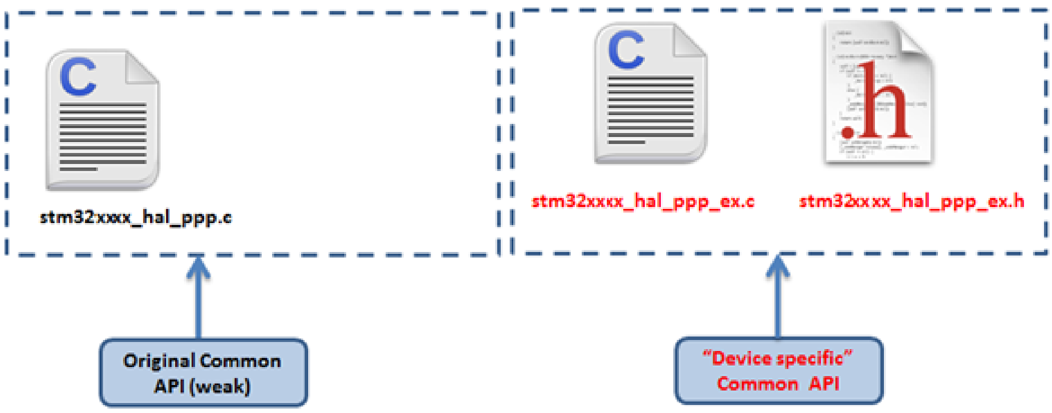


Example: stm32f4xx_hal_adc.c/h

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.

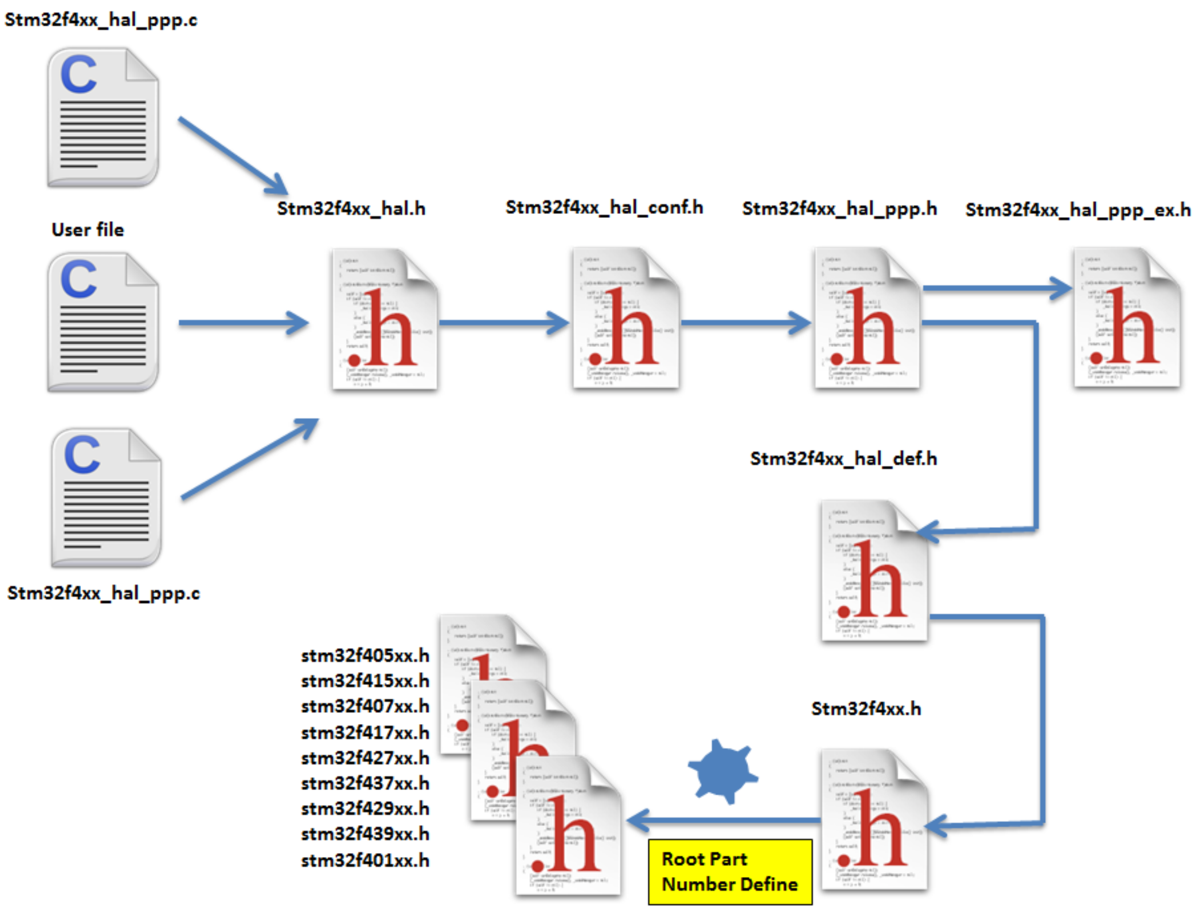
Example:

```
#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
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)

Configuration item	Description	Default Value
EXTERNAL_CLOCK_VALUE	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)
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE
INSTRUCTION_CACHE_ENABLE	Enables instruction cache	TRUE
DATA_CACHE_ENABLE	Enables data cache	TRUE
USE HAL_PPP_MODULE	Enables module to be used in the HAL driver	
MAC_ADDRx	Ethernet peripheral configuration : MAC address	
ETH_RX_BUF_SIZE	Ethernet buffer size for data reception	ETH_MAX_PACKET_SIZE
ETH_TX_BUF_SIZE	Ethernet buffer size for trasmit	ETH_MAX_PACKET_SIZE
ETH_RXBUFNB	The number of Rx buffers of size ETH_RX_BUF_SIZE	4
ETH_TXBUFNB	The number of Tx buffers of size ETH_RX_BUF_SIZE	4
DP83848_PHY_ADDRESS	DB83848 Ethernet PHY Address	0x01
PHY_RESET_DELAY	PHY Reset delay these values are based on a 1 ms SysTick interrupt	0x000000FF
PHY_CONFIG_DELAY	PHY Configuration delay	0x00000FFF
PHY_BCR PHY_BSR	Common PHY Registers	
PHY_SR PHY_MICR PHY_MISR	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 retreiving 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: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input floating – GPIO_MODE_OUTPUT_PP : Output push-pull – GPIO_MODE_OUTPUT_OD : Output open drain – GPIO_MODE_AF_PP : Alternate function push-pull – GPIO_MODE_AF_OD : Alternate function open drain – GPIO_MODE_ANALOG : Analog mode • External Interrupt mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_FAST

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. *Note: 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: #define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */
__HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)	Enables a given EXTI line Example: __HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)	Disables a given EXTI line. Example: __HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_GENERATE_SWIT (__EXTI_LINE__)	Generates a software interrupt for a given EXTI line. Example: __HAL_PVD_EXTI_ GENERATE_SWIT (PWR_EXTI_LINE_PVD)

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 confgured. In this case the DMA interrupt is configured.
  4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine.
  5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

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

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

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel
- __HAL_DMA_DISABLE: disables the specified DMA channel
- __HAL_DMA_GET_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.*

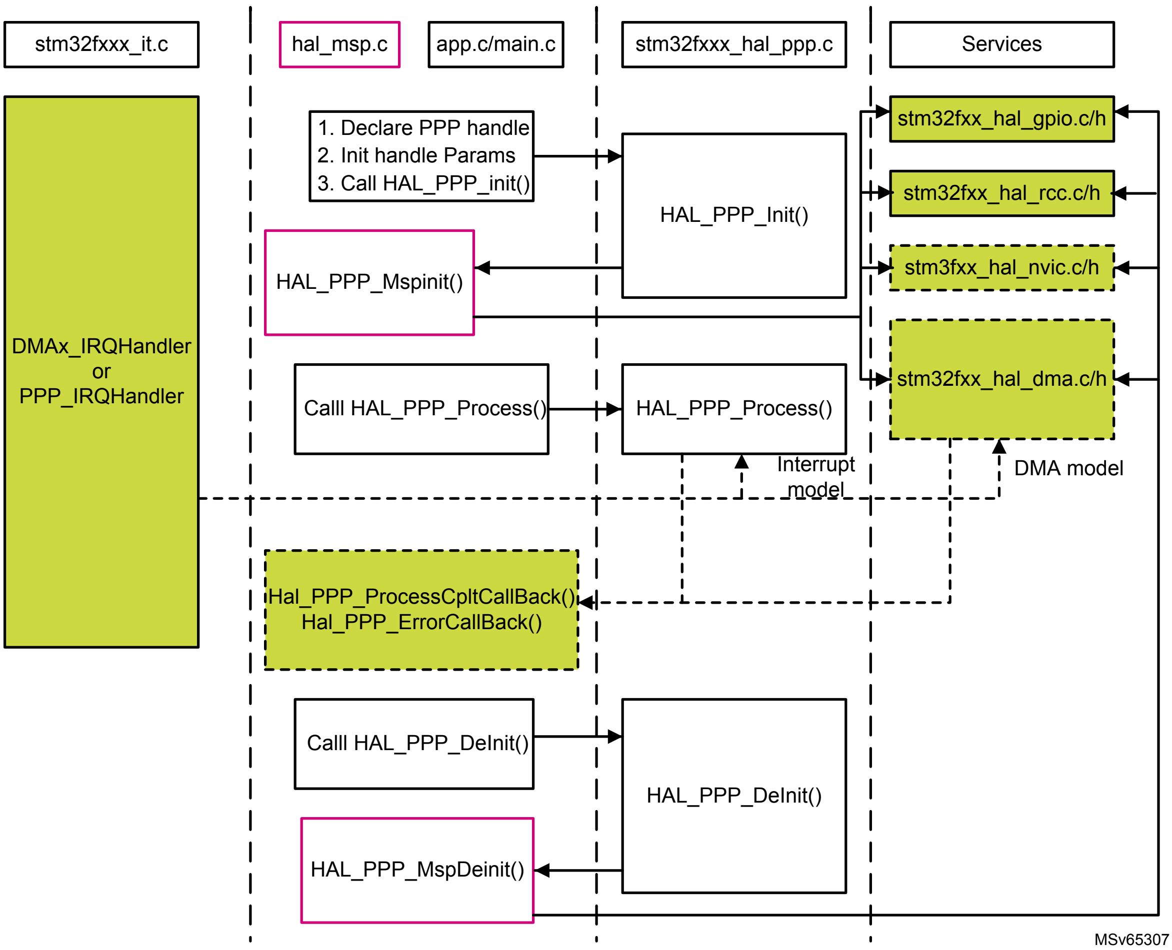
Note: *DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.*

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-Initalizations.
- 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
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

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

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

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

3.12.3 HAL I/O operation process

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

- Polling mode
- Interrupt mode
- DMA mode

3.12.3.1 Polling mode

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

The example below shows the typical Polling mode processing sequence :

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

3.12.3.2 *Interrupt mode*

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

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

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

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *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 = UART3;
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 *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

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) [1]	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 */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

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

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

3.12.4.3 *Run-time checking*

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

To enable the run-time checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in *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
stm32f4xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32f4xx_ll_ppp.h/.c	stm32f4xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are defined within stm32f4xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32f4xx_ll_ppp.h file.
stm32f4xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (such as LL_SYSTICK_xxxxx and LL_LPM_xxxxx "Low Power Mode")
stm32f4xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
stm32f4xx_ll_system.h	System related operations. Example: LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx and LL_VREFBUF_xxx
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

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

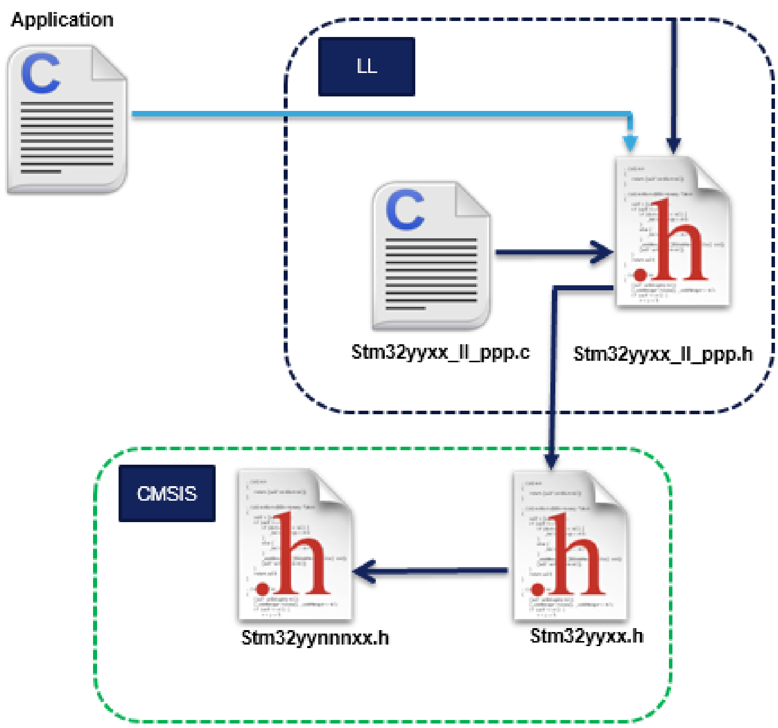
The low-layer files are located in the same HAL driver folder.

Figure 8. Low-layer driver folders

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

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

Figure 9. Low-layer driver CMSIS files



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

4.2 Overview of low-layer APIs and naming rules

4.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in 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	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

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

Table 18. Optional peripheral initialization functions

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

4.2.1.1 Run-time checking

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

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

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

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

4.2.2 Peripheral register-level configuration functions

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

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

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

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

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

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

Table 20. Available function formats

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

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

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

Table 21. Peripheral clock activation/deactivation management

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

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

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

Table 22. Peripheral activation/deactivation management

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

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

Table 23. Peripheral configuration management

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

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

Table 24. Peripheral register management

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

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

5 Cohabiting of HAL and LL

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

5.1 Low-layer driver used in Standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including 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/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.*
2. *When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.*
3. *When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.*

6 HAL System Driver

6.1 HAL Firmware driver API description

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

6.1.1 How to use this driver

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

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

6.1.2 Initialization and Configuration functions

This section provides functions allowing to:

- 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_MspInit

Function name

void HAL_MspInit (void)

Function description

Initialize the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base.

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 paramater "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 trigged 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 tranmit 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 tranmit 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 tranmit 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 + (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 pointeur

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_MspInit

Function name

void HAL_CEC_MspInit (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_TXACKE: 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_RXACKE: 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_TXACKE: 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_RXACKE: 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_TXACKE: 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_RXACKE: 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 (stm32f4xxxx.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_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B

MPU_REGION_SIZE_64B

MPU_REGION_SIZE_128B

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB

MPU_REGION_SIZE_256KB

MPU_REGION_SIZE_512KB

MPU_REGION_SIZE_1MB

MPU_REGION_SIZE_2MB

MPU_REGION_SIZE_4MB

MPU_REGION_SIZE_8MB

MPU_REGION_SIZE_16MB

MPU_REGION_SIZE_32MB

MPU_REGION_SIZE_64MB

MPU_REGION_SIZE_128MB

MPU_REGION_SIZE_256MB

MPU_REGION_SIZE_512MB

MPU_REGION_SIZE_1GB

MPU_REGION_SIZE_2GB

MPU_REGION_SIZE_4GB

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 * pInitVect***
- ***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::pInitVect***
  The initialization vector used also as initialization counter in CTR mode
- ***uint32_t CRYP_ConfigTypeDef::Algorithm***
  DES/ TDES Algorithm ECB/CBC AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t* CRYP_ConfigTypeDef::Header***
  used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.
- ***uint32_t CRYP_ConfigTypeDef::HeaderSize***
  The size of header buffer 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_STATETypeDef 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_STATETypeDef __CRYP_HandleTypeDef::State***
  CRYP peripheral state
- ***__IO uint32_t __CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***uint32_t __CRYP_HandleTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag, used when configuration can be skipped

- ***uint32_t __CRYP_HandleTypeDef::SizesSum***
  Sum of successive payloads lengths (in bytes), stored for a single signature computation after several messages processing

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 parametres, finally user can start encryption/decryption.
5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.
6. To process a single message with consecutive calls to HAL_CRYP_Encrypt() or HAL_CRYP_Decrypt() without having to configure again the Key or the Initialization Vector between each API call, the field KeyIVConfigSkip of the initialization structure must be set to CRYP_KEYIVCONFIG_ONCE. Same is true for consecutive calls of HAL_CRYP_Encrypt_IT(), HAL_CRYP_Decrypt_IT(), HAL_CRYP_Encrypt_DMA() or HAL_CRYP_Decrypt_DMA().

The cryptographic processor supports following standards:

1. The data encryption standard (DES) and Triple-DES (TDES) supported only by CRYP1 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_CRYP_GetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_GetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Get CRYP Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a CRYP_ConfigTypeDef structure
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_Encrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output, uint32_t Timeout)

Function description

Encryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(ciphertext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Decrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output, uint32_t Timeout)

Function description

Decryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(plaintext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Encrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Encryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Decryption in itnterrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_Encrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Encryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Decryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_IRQHandler

Function name

void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)

Function description

This function handles cryptographic interrupt request.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_GetState

Function name

HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)

Function description

Returns the CRYP state.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **HAL:** state

HAL_CRYP_InCpltCallback

Function name

void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Input FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_OutCpltCallback

Function name

void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Output FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_ErrorCallback

Function name

void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)

Function description

CRYP error callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_GetError

Function name

uint32_t HAL_CRYP_GetError (CRYP_HandleTypeDef * hcryp)

Function description

Return the CRYP error code.

Parameters

- **hcryp:** : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for the CRYP 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 CRYP

CRYP

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_CRYP_ERROR_NONE

No error

HAL_CRYP_ERROR_WRITE

Write error

HAL_CRYP_ERROR_READ

Read error

HAL_CRYP_ERROR_DMA

DMA error

HAL_CRYP_ERROR_BUSY

Busy flag error

HAL_CRYP_ERROR_TIMEOUT

Timeout error

HAL_CRYP_ERROR_NOT_SUPPORTED

Not supported mode

HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE

Sequence are not respected only for GCM or CCM

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE

Description:

- Reset CRYP handle state.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_ENABLE

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_DISABLE

CRYP_FLAG_MASK

Description:
- Check whether the specified CRYP status flag is set or not.

Parameters:
- __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 CRYP 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 CRYP 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 (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)

Function description

generate the GCM authentication TAG.

Parameters

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

Return values

- **HAL:** status

HAL_CRYPEx_AESCCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)

Function description

AES CCM Authentication TAG generation.

Parameters

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

Return values

- **HAL:** status

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/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_DAC_RegisterCallback before calling @ref HAL_DAC_DeInit or @ref HAL_DAC_Init function. When The compilation define USE_HAL_DAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

DAC HAL driver macros list

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

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

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

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 unmasks.

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 unmasks.

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_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: 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_OVRRI: Overrun flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: 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_OVRRI

Overrun interrupt flag

DCMI_FLAG_ERRRI

Synchronization error interrupt flag

DCMI_FLAG_VSYNCRI

VSYNC interrupt flag

DCMI_FLAG_LINERI

Line interrupt flag

DCMI_FLAG_FRAMEMI

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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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_dfsdm.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 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 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.
- MspInitCallback : DFSDM channel MSP init callback.
- MspDeInitCallback : 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.
- MspInitCallback : DFSDM filter MSP init callback.
- MspDeInitCallback : 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 MspInit and MspDeInit 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, MspInit or MspDeInit are not null, the init and de-init 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/de-init. In that case first register the MspInit/MspDeInit 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_ChannelInit()***
- ***HAL_DFSDM_ChannelDeInit()***
- ***HAL_DFSDM_ChannelMspInit()***
- ***HAL_DFSDM_ChannelMspDeInit()***

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_ChannelInit

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

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

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelDeInit

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelDeInit (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 ouput clock is system clock

DFSDM_CHANNEL_OUTPUT_CLOCK_AUDIO

Source for ouput 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.

- MspDeInitCallback : DMA2D MspDeInit. (#) 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.

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 recommanded 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 identifer a DMA_HandleTypeDef structure as parameter.
- **pCallback:** pointer to private callbacsk function which has pointer to a DMA_HandleTypeDef structure as parameter.

Return values

- **HAL:** status

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

UnRegister callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifer a HAL_DMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)

Function description

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 FrameBTAAcknowledgeEnable***

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_MspInit() 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_MspInit().

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 applyed 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 MediaInterface***

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::MediaInterface***
  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 then 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 then 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 RxDMABurstLength***
- ***uint32_t TxDMABurstLength***
- ***uint32_t EnhancedDescriptorFormat***
- ***uint32_t DescriptorSkipLength***
- ***uint32_t DMAArbitration***

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_DMARxFrameInfos

ETH_DMARxFrameInfos 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_DMARxFrameInfos::FSRxDesc***
  First Segment Rx Desc
- ***ETH_DMADescTypeDef* ETH_DMARxFrameInfos::LSRxDesc***
  Last Segment Rx Desc
- ***uint32_t ETH_DMARxFrameInfos::SegCount***
  Segment count
- ***uint32_t ETH_DMARxFrameInfos::length***
  Frame length
- ***uint32_t ETH_DMARxFrameInfos::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_DMARxFrameInfos RxFrameInfos***
- ***__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_DMARxFrameInfos ETH_HandleTypeDef::RxFrameInfos***
  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_CHECKSUMOFFLAOD_ENABLE

ETH_CHECKSUMOFFLAOD_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_DMATxDescFlag (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_DMATxDescFlag (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 [70]

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 "GPIOSel" member from GPIO_InitTypeDef structure.
2. Get current Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   – Provide exiting handle as parameter.
   – Provide pointer on EXTI_ConfigTypeDef structure as second parameter.
3. Clear Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   – Provide exiting handle as parameter.
4. Register callback to treat Exti interrupts using HAL_EXTI_RegisterCallback().
   – Provide exiting handle as first parameter.
   – Provide which callback will be registered using one value from EXTI_CallbackIDTypeDef.
   – Provide callback function pointer.
5. Get interrupt pending bit using HAL_EXTI_GetPending().
6. Clear interrupt pending bit using HAL_EXTI_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) pPendingCbfn)

Function description

Register callback for a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **CallbackID:** User callback identifier. This parameter can be one of
  – EXTI_CallbackIDTypeDef values.
- **pPendingCbfn:** function pointer to be stored as callback.

Return values

- **HAL:** Status.

HAL_EXTI_GetHandle

Function name

HAL_StatusTypeDef HAL_EXTI_GetHandle (EXTI_HandleTypeDef * hexti, uint32_t ExtiLine)

Function description

Store line number as handle private field.

Parameters

- **hexti:** Exti handle.
- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.

Return values

- **HAL:** Status.

HAL_EXTI_IRQHandler

Function name

void HAL_EXTI_IRQHandler (EXTI_HandleTypeDef * hexti)

Function description

Handle EXTI interrupt request.

Parameters

- **hexti:** Exti handle.

Return values

- **none.:**

HAL_EXTI_GetPending

Function name

uint32_t HAL_EXTI_GetPending (EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Get interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** 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 GPIOSel

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

EXTI_LINE_6

External interrupt line 6

EXTI_LINE_7

External interrupt line 7

EXTI_LINE_8

External interrupt line 8

EXTI_LINE_9

External interrupt line 9

EXTI_LINE_10

External interrupt line 10

EXTI_LINE_11

External interrupt line 11

EXTI_LINE_12

External interrupt line 12

EXTI_LINE_13

External interrupt line 13

EXTI_LINE_14

External interrupt line 14

EXTI_LINE_15

External interrupt line 15

EXTI_LINE_16

External interrupt line 16 Connected to the PVD Output

EXTI_LINE_17

External interrupt line 17 Connected to the RTC Alarm event

EXTI_LINE_18

External interrupt line 18 Connected to the USB OTG FS Wakeup from suspend event

EXTI_LINE_19

External interrupt line 19 Connected to the Ethernet Wakeup event

EXTI_LINE_20

External interrupt line 20 Connected to the USB OTG HS (configured in FS) Wakeup event

EXTI_LINE_21

External interrupt line 21 Connected to the RTC Tamper and Time Stamp events

EXTI_LINE_22

External interrupt line 22 Connected to the RTC Wakeup event

EXTI Mode

EXTI_MODE_NONE

EXTI_MODE_INTERRUPT

EXTI_MODE_EVENT

EXTI Trigger

EXTI_TRIGGER_NONE

EXTI_TRIGGER_RISING

EXTI_TRIGGER_FALLING

EXTI_TRIGGER_RISING_FALLING

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 RDPLevel***
- ***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::RDPLevel***
  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_PCROP

PCROP 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_PCROPSTATE

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 s 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 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_fmpi2c.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_fmpi2c.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 * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***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 *hfmpi2c, uint32_t ITFlags, uint32_t ITSources)***
  FMPI2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __FMPI2C_HandleTypeDef::hdmatx***
  FMPI2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __FMPI2C_HandleTypeDef::hdmarx***
  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 hfmpi2c;

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 hfmpi2c 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 hfmpi2c 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(&hfmpi2c) 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, functionnal 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 sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  – 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 oposite 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.
- 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_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.
- MspDeInitCallback : callback for Msp DeInit.

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 MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_FMPI2C_Init()/ @ref HAL_FMPI2C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the @ref HAL_FMPI2C_Init()/ @ref HAL_FMPI2C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_FMPI2C_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in @ref HAL_FMPI2C_STATE_READY or @ref HAL_FMPI2C_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_FMPI2C_RegisterCallback() before calling @ref HAL_FMPI2C_DeInit() 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_DeInit() to restore the default configuration of the selected FMPI2Cx peripheral.

This section contains the following APIs:

- *HAL_FMPI2C_Init()*
- *HAL_FMPI2C_DeInit()*
- *HAL_FMPI2C_MspInit()*
- *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_MspInit

Function name

void HAL_FMPI2C_MspInit (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 * hfmpi2c, 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

- **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
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_FMPI2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_FMPI2C_Master_Receive (FMPI2C_HandleTypeDef * hfmpi2c, 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

- **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
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_FMPI2C_Slave_Transmit

Function name

HAL_StatusTypeDef HAL_FMPI2C_Slave_Transmit (FMPI2C_HandleTypeDef * hfmpi2c, 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 * hfmpi2c, 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

- **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_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_FMPI2C_Slave_Transmit_IT (FMPI2C_HandleTypeDef * hfmpi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave 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.
- **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 * hfmpi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave 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.
- **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 * 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 Interrupt 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_IT

Function name

HAL_StatusTypeDef HAL_FMPI2C_Mem_Read_IT (FMPI2C_HandleTypeDef * hfmpi2c, 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

- **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_Master_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_FMPI2C_Master_Seq_Transmit_IT (FMPI2C_HandleTypeDef * hfmpi2c, 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

- **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
- **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 * hfmpi2c, 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

- **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
- **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 * hfmpi2c, 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

- **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
- **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 * hfmpi2c, 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

- **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
- **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 * hfmpi2c)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hfmpi2c:** 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 * hfmpi2c)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hfmpi2c:** 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 * hfmpi2c, uint16_t DevAddress)

Function description

Abort a master FMPI2C IT or DMA process communication 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

Return values

- **HAL:** status

HAL_FMPI2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_FMPI2C_Master_Transmit_DMA (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 DMA.

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_DMA

Function name

HAL_StatusTypeDef HAL_FMPI2C_Master_Receive_DMA (FMPI2C_HandleTypeDef * hfmpi2c, 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

- **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_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 * hfmpi2c, 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

- **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 read

Return values

- **HAL:** status

HAL_FMPI2C_Master_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_FMPI2C_Master_Seq_Transmit_DMA (FMPI2C_HandleTypeDef * hfmpi2c, 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

- **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
- **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 * hfmpi2c, 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

- **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
- **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 * hfmpi2c, 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

- **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
- **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 * hfmpi2c, 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

- **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
- **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 * hfmpi2c)

Function description

This function handles FMPI2C event interrupt request.

Parameters

- **hfmpi2c:** 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 * hfmpi2c)

Function description

This function handles FMPI2C error interrupt request.

Parameters

- **hfmpi2c:** 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 * hfmpi2c)

Function description

Master 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_MasterRxCpltCallback

Function name

void HAL_FMPI2C_MasterRxCpltCallback (FMPI2C_HandleTypeDef * hfmpi2c)

Function description

Master 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_SlaveTxCpltCallback

Function name

void HAL_FMPI2C_SlaveTxCpltCallback (FMPI2C_HandleTypeDef * hfmpi2c)

Function description

Slave 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_SlaveRxCpltCallback

Function name

void HAL_FMPI2C_SlaveRxCpltCallback (FMPI2C_HandleTypeDef * hfmpi2c)

Function description

Slave 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_AddrCallback

Function name

void HAL_FMPI2C_AddrCallback (FMPI2C_HandleTypeDef * hfmpi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hfmpi2c:** 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 * hfmpi2c)

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 * hfmpi2c, uint32_t AnalogFilter)

Function description

Configure FMPI2C Analog noise filter.

Parameters

- **hfmpi2c:** 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 * hfmpi2c, uint32_t DigitalFilter)

Function description

Configure FMPI2C Digital noise filter.

Parameters

- **hfmpi2c:** 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_AFRL and GPIOx_AFRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

Function description

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_Accmlt_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 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.
- 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 * pInBuffer, 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.
- **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

- 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 <= 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 functionss

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 intiates 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_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_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_MspInit()*
- *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_DeInit

Function name

HAL_StatusTypeDef HAL_HCD_DeInit (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 to32K

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_MspInit

Function name

void HAL_HCD_MspInit (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, functionnal is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like @ref HAL_I2C_Master_Seq_Transmit_IT() then @ref HAL_I2C_Master_Seq_Transmit_IT() or @ref HAL_I2C_Master_Seq_Transmit_DMA() then @ref HAL_I2C_Master_Seq_Transmit_DMA())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  - I2C_LAST_FRAME_NO_STOP: Sequential usage (Master only), this option allow to manage a restart condition after several call of the same master sequential interface several times (link with option I2C_FIRST_AND_NEXT_FRAME). Usage can, transfer several bytes one by one using HAL_I2C_Master_Seq_Transmit_IT(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME) or HAL_I2C_Master_Seq_Receive_IT(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME) or HAL_I2C_Master_Seq_Transmit_DMA(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME) or HAL_I2C_Master_Seq_Receive_DMA(option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME). Then usage of this option I2C_LAST_FRAME_NO_STOP at the last Transmit or Receive sequence permit to call the oposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - I2C_OTHER_FRAME: Sequential usage (Master only), this option allow to manage a restart condition after each call of the same master sequential interface. Usage can, transfer several bytes one by one with a restart with slave address between each bytes using HAL_I2C_Master_Seq_Transmit_IT(option I2C_FIRST_FRAME then I2C_OTHER_FRAME) or HAL_I2C_Master_Seq_Receive_IT(option I2C_FIRST_FRAME then I2C_OTHER_FRAME) or HAL_I2C_Master_Seq_Transmit_DMA(option I2C_FIRST_FRAME then I2C_OTHER_FRAME) or HAL_I2C_Master_Seq_Receive_DMA(option I2C_FIRST_FRAME then I2C_OTHER_FRAME). Then usage of this option I2C_OTHER_AND_LAST_FRAME at the last frame to help automatic generation of STOP condition.

- Differents sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Master_Seq_Transmit_IT() or using @ref HAL_I2C_Master_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, @ref HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Master_Seq_Receive_IT() or using @ref HAL_I2C_Master_Seq_Receive_DMA()
    - At reception end of current frame transfer, @ref HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_MasterRxCpltCallback()
  - Abort a master IT or DMA I2C process communication with Interrupt using @ref HAL_I2C_Master_Abort_IT()
    - End of abort process, @ref HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using @ref HAL_I2C_EnableListen_IT() @ref HAL_I2C_DisableListen_IT()
    - When address slave I2C match, @ref HAL_I2C_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end @ref HAL_I2C_ListenCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Slave_Seq_Transmit_IT() or using @ref HAL_I2C_Slave_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, @ref HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using @ref HAL_I2C_Slave_Seq_Receive_IT() or using @ref HAL_I2C_Slave_Seq_Receive_DMA()
    - At reception end of current frame transfer, @ref HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, @ref HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_I2C_ErrorCallback()

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 DeInit/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_DMAPause()*
- *HAL_I2S_DMAResume()*
- *HAL_I2S_DMAStop()*
- *HAL_I2S_IRQHandler()*
- *HAL_I2S_TxHalfCpltCallback()*
- *HAL_I2S_TxCpltCallback()*
- *HAL_I2S_RxHalfCpltCallback()*
- *HAL_I2S_RxCpltCallback()*
- *HAL_I2S_ErrorCallback()*

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_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_I2SEXT_FLUSH_RX_DR macro must be used to flush the remaining data inside I2Sx and I2Sx_ext DR registers and avoid using DeInit/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_MspInit

Function name

void HAL_IRDA_MspInit (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 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_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 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_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 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_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 READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hirda:** 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_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

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_PEFLAG

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 (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode : When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros.

Min-max timeout value @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI frequency dispersion. 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 * hlptim)

Function description

Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_DeInit

Function name

HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize the LPTIM peripheral.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_MspInit

Function name

void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_MspDeInit

Function name

void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM in Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM Set once mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Encoder interface.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Counter mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hlptim)

Function description

Return the current counter value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Autoreload (Period) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Autoreload:** value.

HAL_LPTIM_ReadCompare

Function name

uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Compare (Pulse) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Compare:** value.

HAL_LPTIM_IRQHandler

Function name

void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)

Function description

Handle LPTIM interrupt request.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareMatchCallback

Function name

void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name

void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Trigger detected callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Down to Up callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Up to Down callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)

Function description

Return the LPTIM handle state.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** state

LPTIM_Disable

Function name

void LPTIM_Disable (LPTIM_HandleTypeDef * hlptim)

Function description

Disable LPTIM HW instance.

Parameters

- **hlptim:** pointer to a LPTIM_HandleTypeDef structure that contains the configuration information for LPTIM module.

Return values

- **None:**

Notes

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section.

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_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

Description:

- Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

Notes:

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section. Please call HAL_LPTIM_GetState() after a call to __HAL_LPTIM_DISABLE to check for TIMEOUT.

__HAL_LPTIM_START_CONTINUOUS

Description:

- Start the LPTIM peripheral in Continuous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

Description:

- Start the LPTIM peripheral in single mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_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::FBStartAdress***
  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 serval 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_HandleTypeDfef 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 ManDeflECC***
- ***__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::ManDeflECC***
  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 defaullt 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_MspInit

Function name

void HAL_MMC_MspInit (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 * hmmc)

Function description

MMC error callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_AbortCallback

Function name

void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hmmc)

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 * hmmc, 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 * hmmc)

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 * hmmc, 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 * hmmc, 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 * hmmc, 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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
  – SDIO_IT_RXFIFOF: 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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
  - SDIO_IT_RXFIFOF: 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_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_CMDSENT: 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_TXFIFOF: Transmit FIFO full
  - SDIO_FLAG_RXFIFOF: 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_CMDSENT: 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_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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
  - SDIO_IT_RXFIFOF: 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_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_CMDSENT: 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_RXFIFOF: 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 adressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits adressing or words for 16 bits addressing

- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages
- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory 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 sequnece 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 phusical 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 * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)

Function description

Perform NAND memory Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ComSpace_Timing:** pointer to Common space timing structure
- **AttSpace_Timing:** pointer to Attribute space timing structure

Return values

- **HAL:** status

HAL_NAND_DeInit

Function name

HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)

Function description

Perform NAND memory De-Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ConfigDevice

Function name

HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, NAND_DeviceConfigTypeDef * pDeviceConfig)

Function description

Configure the device: Enter the physical parameters of the device.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pDeviceConfig:** pointer to NAND_DeviceConfigTypeDef structure

Return values

- **HAL:** status

HAL_NAND_Read_ID

Function name

HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)

Function description

Read the NAND memory electronic signature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values

- **HAL:** status

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP Init.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_MspDeInit

Function name

void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP DeInit.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

This function handles NAND device interrupt request.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

NAND interrupt feature callback.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_Reset

Function name

HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)

Function description

NAND memory reset.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_Read_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Read_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Erase_Block

Function name

HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

NAND memory Block erase.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **HAL:** status

HAL_NAND_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.

- MspDeInitCallback : NOR MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_NOR_Init and if the state is HAL_NOR_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_NOR_Init and @ref HAL_NOR_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_NOR_Init and @ref HAL_NOR_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_NOR_RegisterCallback before calling @ref HAL_NOR_DeInit or @ref HAL_NOR_Init function. When The compilation define USE_HAL_NOR_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

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_DeInit()*
- *HAL_NOR_MspInit()*
- *HAL_NOR_MspDeInit()*
- *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_MspInit

Function name

void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_MspDeInit

Function name

void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_Start

Function name

HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)

Function description

Start the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Stop

Function name

HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function description

Stop the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_IRQHandler

Function name

void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)

Function description

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 in entered with WFI or WFE instruction. This parameter can be one of the following values:
  – PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction
  – PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction

Return values

- **None:**

Notes

- In Stop 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 in 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 in 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 perioed expires.
- MspInitCallback : QSPI MspInit.
- MspDeInitCallback : QSPI MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function @ref HAL_QSPI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.

- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : QSPI MspInit.
- MspDeInitCallback : QSPI MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the @ref HAL_QSPI_Init and if the state is HAL_QSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_QSPI_Init and @ref HAL_QSPI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the @ref HAL_QSPI_Init and @ref HAL_QSPI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_QSPI_RegisterCallback before calling @ref HAL_QSPI_DeInit or @ref HAL_QSPI_Init function. When The compilation define USE_HAL_QSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Workarounds linked to Silicon Limitation

1. Workarounds Implemented inside HAL Driver
   – Extra data written in the FIFO at the end of a read transfer

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 HSICalibrationValue***
- ***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::HSICalibrationValue***
  The HSI calibration trimming value (default is RCC_HSICALIBRATION_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 sourceHSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO1 and MCO2 OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

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_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
- **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_LPWRRST

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_HSICALIBRATION_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_HSICALIBRATION_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

PLLP 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_EnablePLLI2S

Function name

HAL_StatusTypeDef HAL_RCCEx_EnablePLLI2S (RCC_PLLI2SInitTypeDef * PLLI2SInit)

Function description

Enable PLLI2S.

Parameters

- **PLLI2SInit:** pointer to an RCC_PLLI2SInitTypeDef structure that contains the configuration information for the PLLI2S

Return values

- **HAL:** status

HAL_RCCEx_DisablePLLI2S

Function name

HAL_StatusTypeDef HAL_RCCEx_DisablePLLI2S (void)

Function description

Disable PLLI2S.

Return values

- **HAL:** status

HAL_RCCEx_EnablePLLSAI

Function name

HAL_StatusTypeDef HAL_RCCEx_EnablePLLSAI (RCC_PLLSAIInitTypeDef * PLLSAIInit)

Function description

Enable PLLSAI.

Parameters

- **PLLSAIInit:** pointer to an RCC_PLLSAIInitTypeDef structure that contains the configuration information for the PLLSAI

Return values

- **HAL:** status

HAL_RCCEx_DisablePLLSAI

Function name

HAL_StatusTypeDef HAL_RCCEx_DisablePLLSAI (void)

Function description

Disable PLLSAI.

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_DCMI_CLK_ENABLE

__HAL_RCC_DCMI_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_DCMI_IS_CLK_ENABLED

__HAL_RCC_DCMI_IS_CLK_DISABLED

__HAL_RCC_CRYP_IS_CLK_ENABLED

__HAL_RCC_CRYP_IS_CLK_DISABLED

__HAL_RCC_HASH_IS_CLK_ENABLED

__HAL_RCC_HASH_IS_CLK_DISABLED

__HAL_RCC_USB_OTG_FS_IS_CLK_ENABLED

__HAL_RCC_USB_OTG_FS_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_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.
- __PLLP__: 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(PLLI2SQ) / __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(PLLSAIQ) / __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(PLLSAIR) / __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_PERIPHCLOCK

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_SDIOCLKSOURCE_CLK48

RCC_SDIOCLKSOURCE_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: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until current date is read.

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_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:12] are don't care in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDMASK_SS14

SS[14] is don't care in Alarm comparison.Only SS[13:0] are compared

RTC_ALARMSUBSECONDMASK_NONE

SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC 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_RSF

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_ASYNCH_PREDIV

IS_RTC_SYNCH_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_TAFCR 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_TAFCR 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 SmouthCalibMinusPulsesValue)

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 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.
- **SmouthCalibMinusPulsesValue:** Select the value of CALM[80] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmouthCalibMinusPulsesValue 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 2exp20 RTCCLK seconds

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK seconds

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK seconds

RTC Smooth Calib Plus Pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

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_TRIState_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_FrameInitTypeDef

SAI_FrameInitTypeDef 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_FrameInitTypeDef::FrameLength***
  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.
  Note:
  – If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***
  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***
  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***
- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***
  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***
- ***uint32_t SAI_FrameInitTypeDef::FSOffset***
  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

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 * pBuffPtr***
- ***uint16_t XferSize***
- ***uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***SAIcallback mutecallback***
- ***void(* InterruptServiceRoutine***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SAI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size

- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object
- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code

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, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: *The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.*

Note: *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_MspInit()*
- *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_DMAPause()*
- *HAL_SAI_DMAResume()*
- *HAL_SAI_DMAStop()*
- *HAL_SAI_Abort()*
- *HAL_SAI_Transmit_DMA()*
- *HAL_SAI_Receive_DMA()*
- *HAL_SAI_EnableTxMuteMode()*

- *HAL_SAI_DisableTxMuteMode()*
- *HAL_SAI_EnableRxMuteMode()*
- *HAL_SAI_DisableRxMuteMode()*
- *HAL_SAI_IRQHandler()*
- *HAL_SAI_TxCpltCallback()*
- *HAL_SAI_TxHalfCpltCallback()*
- *HAL_SAI_RxCpltCallback()*
- *HAL_SAI_RxHalfCpltCallback()*
- *HAL_SAI_ErrorCallback()*

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 FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **protocol:** one of the supported protocol SAI Supported protocol
- **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module.
- **nbslot:** Number of slot.

Return values

- **HAL:** status

HAL_SAI_Init

Function name

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI according to the specified parameters.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DeInit

Function name

HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI peripheral.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_MspInit

Function name

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_MspDeInit

Function name

void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_Transmit

Function name

HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Receive

Function name

HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_DMAPause

Function name

HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)

Function description

Pause the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAResume

Function name

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

Resume the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAStop

Function name

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

Stop the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_Abort

Function name

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

Abort the current transfer and disable the SAI.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

Enable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **val:** value sent during the mute SAI Block Mute Value

Return values

- **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)

Function description

Enable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback:** function called when the mute is detected.
- **counter:** number a data before mute detection max 63.

Return values

- **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_IRQHandler

Function name

void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)

Function description

Handle SAI interrupt request.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer Half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_ErrorCallback

Function name

void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)

Function description

SAI error callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_GetState

Function name

HAL_SAI_StateTypeDef HAL_SAI_GetState (SAI_HandleTypeDef * hsai)

Function description

Return the SAI handle state.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** state

HAL_SAI_GetError

Function name

uint32_t HAL_SAI_GetError (SAI_HandleTypeDef * hsai)

Function description

Return the SAI error code.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- **SAI:** Error Code

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_MUTEDET: Mute detection interrupt enable
  – SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  – SAI_IT_FREQ: FIFO request interrupt enable
  – SAI_IT_CNRDY: Codec not ready interrupt enable
  – SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  – SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

__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 TRIState 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 ManDeflECC*
- *__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.
- MspInitCallback : SD MspInit.

- 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_MspInit

Function name

void HAL_SD_MspInit (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 rmode

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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
  - SDIO_IT_RXFIFOF: 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_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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
  – SDIO_IT_RXFIFOF: 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_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_CMDSENT: 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_TXFIFOF: Transmit FIFO full
  – SDIO_FLAG_RXFIFOF: 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_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_CMDSENT: 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_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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
  - SDIO_IT_RXFIFOF: 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_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_CMDSENT: 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_MspInit

Function name

void HAL_SDRAM_MspInit (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:
  – IntegerDivider = ((PCLKx) / (16 * (hsc->Init.BaudRate)))
  – FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 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 tranmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

(#) 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 tranmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:

- *HAL_SMARTCARD_Transmit()*
- *HAL_SMARTCARD_Receive()*
- *HAL_SMARTCARD_Transmit_IT()*
- *HAL_SMARTCARD_Receive_IT()*
- *HAL_SMARTCARD_Transmit_DMA()*
- *HAL_SMARTCARD_Receive_DMA()*
- *HAL_SMARTCARD_Abort()*
- *HAL_SMARTCARD_AbortTransmit()*
- *HAL_SMARTCARD_AbortReceive()*
- *HAL_SMARTCARD_Abort_IT()*
- *HAL_SMARTCARD_AbortTransmit_IT()*
- *HAL_SMARTCARD_AbortReceive_IT()*
- *HAL_SMARTCARD_IRQHandler()*
- *HAL_SMARTCARD_TxCpltCallback()*
- *HAL_SMARTCARD_RxCpltCallback()*
- *HAL_SMARTCARD_ErrorCallback()*

- *HAL_SMARTCARD_AbortCpltCallback()*
- *HAL_SMARTCARD_AbortTransmitCpltCallback()*
- *HAL_SMARTCARD_AbortReceiveCpltCallback()*

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 Periphal 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_BTF: 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 CtrlRx/DataRx with the required parameters.
      - Configure the DMA Channel.
      - Associate the initialized DMA handle to the SPDIFRX DMA CtrlRx/DataRx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CtrlRx/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_SYNCD

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_SYNCDIE

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_MspInit()*
- *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_DMAPause()*
- *HAL_SPI_DMAResume()*
- *HAL_SPI_DMAStop()*
- *HAL_SPI_IRQHandler()*
- *HAL_SPI_TxCpltCallback()*
- *HAL_SPI_RxCpltCallback()*
- *HAL_SPI_TxRxCpltCallback()*

- *HAL_SPI_TxHalfCpltCallback()*
- *HAL_SPI_RxHalfCpltCallback()*
- *HAL_SPI_TxRxHalfCpltCallback()*
- *HAL_SPI_ErrorCallback()*
- *HAL_SPI_AbortCpltCallback()*

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 ChannelNState***
- ***__IO HAL_TIM_DMABurstStateTypeDef DMABurstState***

Field Documentation

- ***TIM_TypeDef* TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a ***DMA_Handle_index***

- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
  TIM operation state
- ***__IO HAL_TIM_ChannelStateTypeDef TIM_HandleTypeDef::ChannelState[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 TIx input, without taking in account the comparison.

HAL_TIM_ConfigOCrefClear

Function name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function description

Configures the OCRef clear feature.

Parameters

- **htim:** TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4

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 TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** The Input Polarity. This parameter can be one of the following values:
  – TIM_ICPOLARITY_RISING
  – TIM_ICPOLARITY_FALLING
  – TIM_ICPOLARITY_BOTHEDGE
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
  – TIM_ICSELECTION_DIRECTTI: TIM Input 1 is selected to be connected to IC1.
  – TIM_ICSELECTION_INDIRECTTI: TIM Input 1 is selected to be connected to IC2.
  – TIM_ICSELECTION_TRC: TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values

- **None:**

Notes

- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name

void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)

Function description

Timer Output Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The ouput configuration structure

Return values

- **None:**

TIM_ETR_SetConfig

Function name

void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function description

Configures the TIMx External Trigger (ETR).

Parameters

- **TIMx:** to select the TIM peripheral
- **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values:
  – TIM_ETRPRESCALER_DIV1: ETRP Prescaler OFF.
  – TIM_ETRPRESCALER_DIV2: ETRP frequency divided by 2.
  – TIM_ETRPRESCALER_DIV4: ETRP frequency divided by 4.
  – TIM_ETRPRESCALER_DIV8: ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values:
  – TIM_ETRPOLARITY_INVERTED: active low or falling edge active.
  – TIM_ETRPOLARITY_NONINVERTED: active high or rising edge active.
- **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values

- **None:**

TIM_DMADelayPulseHalfCplt

Function name

void TIM_DMADelayPulseHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Delay Pulse half complete callback.

Parameters

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

Return values

- **None:**

TIM_DMAError

Function name

void TIM_DMAError (DMA_HandleTypeDef * hdma)

Function description

TIM DMA error callback.

Parameters

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

Return values

- **None:**

TIM_DMACaptureCplt

Function name

void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture complete callback.

Parameters

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

Return values

- **None:**

TIM_DMACaptureHalfCplt

Function name

void TIM_DMACaptureHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture half complete callback.

Parameters

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

Return values

- **None:**

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

Enables or disables the TIM Capture Compare Channel x.

Parameters

- **TIMx:** to select the TIM peripheral
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1
  – TIM_CHANNEL_2: TIM Channel 2
  – TIM_CHANNEL_3: TIM Channel 3
  – TIM_CHANNEL_4: TIM Channel 4
- **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 TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2

External clock source mode 2

TIM_CLOCKSOURCE_INTERNAL

Internal clock source

TIM_CLOCKSOURCE_ITR0

External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1

External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2

External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3

External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_TI1ED

External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1

External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2

External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ETRMODE1

External clock source mode 1 (ETRF)

TIM 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 trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_2TRANSFERS

The transfer is done to 2 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_3TRANSFERS

The transfer is done to 3 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_4TRANSFERS

The transfer is done to 4 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_5TRANSFERS

The transfer is done to 5 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_6TRANSFERS

The transfer is done to 6 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_7TRANSFERS

The transfer is done to 7 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_8TRANSFERS

The transfer is done to 8 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_9TRANSFERS

The transfer is done to 9 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_10TRANSFERS

The transfer is done to 10 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_11TRANSFERS

The transfer is done to 11 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_12TRANSFERS

The transfer is done to 12 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_13TRANSFERS

The transfer is done to 13 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_14TRANSFERS

The transfer is done to 14 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_15TRANSFERS

The transfer is done to 15 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_16TRANSFERS

The transfer is done to 16 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_17TRANSFERS

The transfer is done to 17 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_18TRANSFERS

The transfer is done to 18 registers starting trom TIMx_CR1 + TIMx_DCR.DBA

TIM DMA Sources

TIM_DMA_UPDATE

DMA request is triggered by the update event

TIM_DMA_CC1

DMA request is triggered by the capture/compare macth 1 event

TIM_DMA_CC2

DMA request is triggered by the capture/compare macth 2 event event

TIM_DMA_CC3

DMA request is triggered by the capture/compare macth 3 event event

TIM_DMA_CC4

DMA request is triggered by the capture/compare macth 4 event event

TIM_DMA_COM

DMA request is triggered by the commutation event

TIM_DMA_TRIGGER

DMA request is triggered by the trigger event

TIM Encoder Input Polarity

TIM_ENCODERINPUTPOLARITY_RISING

Encoder input with rising edge polarity

TIM_ENCODERINPUTPOLARITY_FALLING

Encoder input with falling edge polarity

TIM Encoder Mode

TIM_ENCODERMODE_TI1

Quadrature encoder mode 1, x2 mode, counts up/down on TI1FP1 edge depending on TI2FP2 level

TIM_ENCODERMODE_TI2

Quadrature encoder mode 2, x2 mode, counts up/down on TI2FP2 edge depending on TI1FP1 level.

TIM_ENCODERMODE_TI12

Quadrature encoder mode 3, x4 mode, counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

TIM 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 TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

Update interrupt

TIM_IT_CC1

Capture/Compare 1 interrupt

TIM_IT_CC2

Capture/Compare 2 interrupt

TIM_IT_CC3

Capture/Compare 3 interrupt

TIM_IT_CC4

Capture/Compare 4 interrupt

TIM_IT_COM

Commutation interrupt

TIM_IT_TRIGGER

Trigger interrupt

TIM_IT_BREAK

Break interrupt

TIM 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_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   – Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   – Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   – Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

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_OCN_Start()*
- *HAL_TIMEx_OCN_Stop()*
- *HAL_TIMEx_OCN_Start_IT()*
- *HAL_TIMEx_OCN_Stop_IT()*
- *HAL_TIMEx_OCN_Start_DMA()*

- *HAL_TIMEx_OCN_Stop_DMA()*

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 TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with interrupt.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_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 TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with DMA.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_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 TI1 of the Interface Timer detect a commutation at its input TI1.
- The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_MasterConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)

Function description

Configures the TIM in master mode.

Parameters

- **htim:** TIM handle.
- **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description

Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

Parameters

- **htim:** TIM handle
- **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- **HAL:** status

Notes

- Interrupts can be generated when an active level is detected on the break input, the break 2 input or the system break input. Break interrupt can be enabled by calling the __HAL_TIM_ENABLE_IT macro.

HAL_TIMEx_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_GetChannelNState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIMEx_GetChannelNState (TIM_HandleTypeDef * htim, uint32_t ChannelN)

Function description

Return actual state of the TIM complementary channel.

Parameters

- **htim:** TIM handle
- **ChannelN:** TIM Complementary channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3

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 * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef gState***
- ***__IO HAL_UART_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***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::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code

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_DMAPause()
- Resume the DMA Transfer using HAL_UART_DMAResume()
- Stop the DMA Transfer using HAL_UART_DMAStop()

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_ReceiveToIdle()
4. Non-Blocking mode API with Interrupt: (+) HAL_UARTEx_ReceiveToIdle_IT()
5. Non-Blocking mode API with DMA: (+) HAL_UARTEx_ReceiveToIdle_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_ReceiveToIdle() (#) Non-Blocking mode API with Interrupt:
- HAL_UARTEx_ReceiveToIdle_IT() (#) Non-Blocking mode API with DMA:
- HAL_UARTEx_ReceiveToIdle_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_DMAPause()***
- ***HAL_UART_DMAResume()***
- ***HAL_UART_DMAStop()***
- ***HAL_UARTEx_ReceiveToIdle()***
- ***HAL_UARTEx_ReceiveToIdle_IT()***
- ***HAL_UARTEx_ReceiveToIdle_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_UARTEx_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_ReceiveToIdle

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle (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_ReceiveToIdle_IT

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_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_ReceiveToIdle_DMA

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_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_UARTEx_RxEventCallback

Function name

void HAL_UARTEx_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_PEFLAG

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_DMAPause()
- 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_PEFLAG

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 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 triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.

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 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=0xFFF

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=0xFFF

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=0xFFF

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=0xFFF

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=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData10
- JDR2 JDATA LL_ADC_INJ_ReadConversionData10
- JDR3 JDATA LL_ADC_INJ_ReadConversionData10
- JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData8
- JDR2 JDATA LL_ADC_INJ_ReadConversionData8
- JDR3 JDATA LL_ADC_INJ_ReadConversionData8
- JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_INJ_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData6 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData6
- JDR2 JDATA LL_ADC_INJ_ReadConversionData6
- JDR3 JDATA LL_ADC_INJ_ReadConversionData6
- JDR4 JDATA LL_ADC_INJ_ReadConversionData6

LL_ADC_IsActiveFlag_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 JEOC 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 JEOC2 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 JEOC3 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 (aligment 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=0xFFF

Return value:

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

Notes:

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

__LL_ADC_COMMON_INSTANCE

Description:

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

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

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

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

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

Parameters:

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

Return value:

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

Notes:

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

__LL_ADC_DIGITAL_SCALE

Description:

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

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data 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 to 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 OTGHSEN 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 OTGHSEN 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 OTGHSEN 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 ETHMACPTPLPEN 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 Periphs)

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 ETHMACPTPLPEN 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 CRYPRST 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 CRYPRST 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 CRYPLPEN LL_AHB2_GRP1_EnableClockLowPower
- AHB2LPENR AESLPEN LL_AHB2_GRP1_EnableClockLowPower
- AHB2LPENR HASHLPEN LL_AHB2_GRP1_EnableClockLowPower
- AHB2LPENR RNGLPEN 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 RNGLPEN 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 FMCRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR FSMCRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR QSPIRST LL_AHB3_GRP1_ForceReset

LL_AHB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_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 FMCRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR FSMCRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR QSPIRST 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 SPDIFRXLPEN 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 DACLPEN 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 SPDIFRXLPEN 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 rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_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_HARDFAULT_NMI

Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers

LL_MPU_CTRL_PRIVILEGED_DEFAULT

Enable privileged software access to default memory map

LL_MPU_CTRL_HFNMI_PRIVDEF

Enable NMI and privileged SW access

Handler Fault type

LL_HANDLER_FAULT_USG

Usage fault

LL_HANDLER_FAULT_BUS

Bus fault

LL_HANDLER_FAULT_MEM

Memory management fault

MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE

Instruction fetches enabled

LL_MPU_INSTRUCTION_ACCESS_DISABLE

Instruction fetches disabled

MPU Region Number

LL_MPU_REGION_NUMBER0

REGION Number 0

LL_MPU_REGION_NUMBER1

REGION Number 1

LL_MPU_REGION_NUMBER2

REGION Number 2

LL_MPU_REGION_NUMBER3

REGION Number 3

LL_MPU_REGION_NUMBER4

REGION Number 4

LL_MPU_REGION_NUMBER5

REGION Number 5

LL_MPU_REGION_NUMBER6

REGION Number 6

LL_MPU_REGION_NUMBER7

REGION Number 7

MPU Region Privileges

LL_MPU_REGION_NO_ACCESS

No access

LL_MPU_REGION_PRIV_RW

RW privileged (privileged access only)

LL_MPU_REGION_PRIV_RW_URO

RW privileged - RO user (Write in a user program generates a fault)

LL_MPU_REGION_FULL_ACCESS
RW privileged & user (Full access)

LL_MPU_REGION_PRIV_RO
RO privileged (privileged read only)

LL_MPU_REGION_PRIV_RO_URO
RO privileged & user (read only)

MPU Region Size

LL_MPU_REGION_SIZE_32B
32B Size of the MPU protection region

LL_MPU_REGION_SIZE_64B
64B Size of the MPU protection region

LL_MPU_REGION_SIZE_128B
128B Size of the MPU protection region

LL_MPU_REGION_SIZE_256B
256B Size of the MPU protection region

LL_MPU_REGION_SIZE_512B
512B Size of the MPU protection region

LL_MPU_REGION_SIZE_1KB
1KB Size of the MPU protection region

LL_MPU_REGION_SIZE_2KB
2KB Size of the MPU protection region

LL_MPU_REGION_SIZE_4KB
4KB Size of the MPU protection region

LL_MPU_REGION_SIZE_8KB
8KB Size of the MPU protection region

LL_MPU_REGION_SIZE_16KB
16KB Size of the MPU protection region

LL_MPU_REGION_SIZE_32KB
32KB Size of the MPU protection region

LL_MPU_REGION_SIZE_64KB
64KB Size of the MPU protection region

LL_MPU_REGION_SIZE_128KB
128KB Size of the MPU protection region

LL_MPU_REGION_SIZE_256KB
256KB Size of the MPU protection region

LL_MPU_REGION_SIZE_512KB
512KB Size of the MPU protection region

LL_MPU_REGION_SIZE_1MB
1MB Size of the MPU protection region

LL_MPU_REGION_SIZE_2MB

2MB Size of the MPU protection region

LL_MPU_REGION_SIZE_4MB

4MB Size of the MPU protection region

LL_MPU_REGION_SIZE_8MB

8MB Size of the MPU protection region

LL_MPU_REGION_SIZE_16MB

16MB Size of the MPU protection region

LL_MPU_REGION_SIZE_32MB

32MB Size of the MPU protection region

LL_MPU_REGION_SIZE_64MB

64MB Size of the MPU protection region

LL_MPU_REGION_SIZE_128MB

128MB Size of the MPU protection region

LL_MPU_REGION_SIZE_256MB

256MB Size of the MPU protection region

LL_MPU_REGION_SIZE_512MB

512MB Size of the MPU protection region

LL_MPU_REGION_SIZE_1GB

1GB Size of the MPU protection region

LL_MPU_REGION_SIZE_2GB

2GB Size of the MPU protection region

LL_MPU_REGION_SIZE_4GB

4GB Size of the MPU protection region

MPU Shareable Access

LL_MPU_ACCESS_SHAREABLE

Shareable memory attribute

LL_MPU_ACCESS_NOT_SHAREABLE

Not Shareable memory attribute

MPU TEX Level

LL_MPU_TEX_LEVEL0

b000 for TEX bits

LL_MPU_TEX_LEVEL1

b001 for TEX bits

LL_MPU_TEX_LEVEL2

b010 for TEX bits

LL_MPU_TEX_LEVEL4

b100 for TEX bits

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 of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_EnableTrigger
- CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name

__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all 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=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned
- DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all devices. Refer to device datasheet for channels availability.
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned
- DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2 (1)

  (1) On this STM32 serie, parameter not available on all 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 transfered.
  – 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:

- FGMAR 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:

- FGMAR 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=0x3FF

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=0x3FF

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:

- FGCOLR 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:

- FGCOLR 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:

- FGCOLR 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:

- FGCOLR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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=0x3FF

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:

- BGCOLR RED LL_DMA2D_BGND_SetColor
- BGCOLR GREEN LL_DMA2D_BGND_SetColor
- BGCOLR 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:

- BGCOLR 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:

- BGCOLR 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:

- BGCOLR 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:

- BGCOLR 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:

- BGCOLR 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:

- BGCOLR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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:

- BGPFCCR 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 interrup 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 nterrup 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 interrup 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 interrup 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 < fifo_level < 1/4

LL_DMA_FIFOSTATUS_25_50

1/4 < fifo_level < 1/2

LL_DMA_FIFOSTATUS_50_75

1/2 < fifo_level < 3/4

LL_DMA_FIFOSTATUS_75_100

3/4 < fifo_level < 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_fmpi2c.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 = 0x0FThis 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 = 0x3FFThis 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_OWNADDRESS1***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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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 drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_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_FMPSMBUS_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 drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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=0xFFF.
- **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_FMPSMBUS_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/ orTimeoutB).

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=0xFFF.

Return values

- **None:**

Notes

- Macro IS_FMPSMBUS_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=0xFFF

Notes

- Macro IS_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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=0xFFF.

Return values

- **None:**

Notes

- Macro IS_FMPSMBUS_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 Extented Cumulative Clock TimeoutB setting.

Parameters

- **FMPI2Cx:** FMPI2C Instance.

Return values

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

Notes

- Macro IS_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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 ADDRCF 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 NACKCF 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 OVRCF 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_FMPSMBUS_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_FMPSMBUS_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 TIMOUTCF 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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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_FMPSMBUS_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:

- PECR 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_TIMOUTCF

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_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- 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_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- (*): Available in some devices
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- 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_PUSHPULL

Select push-pull as output type

LL_GPIO_OUTPUT_OPENDRAIN

Select open-drain as output type

PIN

LL_GPIO_PIN_0

Select pin 0

LL_GPIO_PIN_1

Select pin 1

LL_GPIO_PIN_2

Select pin 2

LL_GPIO_PIN_3

Select pin 3

LL_GPIO_PIN_4

Select pin 4

LL_GPIO_PIN_5

Select pin 5

LL_GPIO_PIN_6
Select pin 6

LL_GPIO_PIN_7
Select pin 7

LL_GPIO_PIN_8
Select pin 8

LL_GPIO_PIN_9
Select pin 9

LL_GPIO_PIN_10
Select pin 10

LL_GPIO_PIN_11
Select pin 11

LL_GPIO_PIN_12
Select pin 12

LL_GPIO_PIN_13
Select pin 13

LL_GPIO_PIN_14
Select pin 14

LL_GPIO_PIN_15
Select pin 15

LL_GPIO_PIN_ALL
Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO
Select I/O no pull

LL_GPIO_PULL_UP
Select I/O pull up

LL_GPIO_PULL_DOWN
Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW
Select I/O low output speed

LL_GPIO_SPEED_FREQ_MEDIUM
Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH
Select I/O fast output speed

LL_GPIO_SPEED_FREQ_VERY_HIGH
Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:
- Write a value in GPIO register.

Parameters:
- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_GPIO_ReadReg

Description:
- Read a value in GPIO register.

Parameters:
- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:
- Register: value

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 = 0x0FThis feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***
  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FFThis feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***
  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of *I2C_LL_EC_I2C_ACKNOWLEDGE*This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***
  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of *I2C_LL_EC_OWNADDRESS1*This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

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=0xFFF, 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=0xFFF, 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 drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 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 drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 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 Generall 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_EnableSMBusPECCompare

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (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=0xFFF, 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=0xFFF.

__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=0xFFF

__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 be not be prescaled.

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (LPTIM_TypeDef * LPTIMx)

Function description

Get actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 1 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT1_SRC_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

Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_EnableIT_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare match interrupt (CMPMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_EnableIT_CMPM

LL_LPTIM_DisableIT_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare match interrupt (CMPMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_DisableIT_CMPM

LL_LPTIM_IsEnabledIT_CMPM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare match interrupt (CMPMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_IsEnabledIT_CMPM

LL_LPTIM_EnableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload match interrupt (ARRMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Enable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Disable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_EnableIT_CMPOK

LL_LPTIM_DisableIT_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_DisableIT_CMPOK

LL_LPTIM_IsEnabledIT_CMPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMPOKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_IsEnabledIT_CMPOK

LL_LPTIM_EnableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

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 Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR

The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CMP registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE

The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CMP registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM

LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINOUS or SINGLE

LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1

Prescaler division factor is set to 1

LL_LPTIM_PRESCALER_DIV2

Prescaler division factor is set to 2

LL_LPTIM_PRESCALER_DIV4

Prescaler division factor is set to 4

LL_LPTIM_PRESCALER_DIV8

Prescaler division factor is set to 8

LL_LPTIM_PRESCALER_DIV16

Prescaler division factor is set to 16

LL_LPTIM_PRESCALER_DIV32

Prescaler division factor is set to 32

LL_LPTIM_PRESCALER_DIV64

Prescaler division factor is set to 64

LL_LPTIM_PRESCALER_DIV128

Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE

Any trigger active level change is considered as a valid trigger

LL_LPTIM_TRIG_FILTER_2

Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_4

Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_8

Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING

LPTIM counter starts when a rising edge is detected

LL_LPTIM_TRIG_POLARITY_FALLING

LPTIM counter starts when a falling edge is detected

LL_LPTIM_TRIG_POLARITY_RISING_FALLING

LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO

External input trigger is connected to TIMx_ETR input

LL_LPTIM_TRIG_SOURCE_RTCALARMA

External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB

External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1

External input trigger is connected to RTC Tamper 1

LL_LPTIM_TRIG_SOURCE_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 LPlVDS 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 LSERDY 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_SYSCLK

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:

- CFGR 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
- **PLLP_R:** 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/PLLP 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
- **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
- **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/PLLR 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

- **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
  - 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

- **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
  - LL_RCC_PLLSAIM_DIV_52
  - LL_RCC_PLLSAIM_DIV_53

- **PLLN:** Between 50 and 432
- **PLLP:** 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/PLLP 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

- **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
  – LL_RCC_PLLSAIM_DIV_52
  – LL_RCC_PLLSAIM_DIV_53

- **PLLN:** Between 49/50(*) and 432
- **PLLR:** 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/PLLR 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_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC flag Low Power reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_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 LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)

Function description

Disable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- 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 sourceHSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)

Function description

Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_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_GetLTDCClockFreq

Function name

uint32_t LL_RCC_GetLTDCClockFreq (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 DSIxSource)

Function description

Return DSI clock frequency.

Parameters

- **DSIxSource:** 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_LPWRRSTF

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 (PLLP)

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 (PLLR)

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)
- __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_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 (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

__LL_RCC_CALC_PCLK2_FREQ

Description:
- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:
- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:
- PCLK2: clock frequency (in Hz)

Common Write and read registers Macros

LL_RCC_WriteReg

Description:
- Write a value in RCC register.

Parameters:
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_RCC_ReadReg

Description:
- Read a value in RCC register.

Parameters:
- __REG__: Register to be read

Return value:
- Register: value

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

Return32-bit Random Number value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Generated:** 32-bit random value

Reference Manual to LL API cross reference:

- DR RNDATA LL_RNG_ReadRandData32

LL_RNG_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 = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

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 = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

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 (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).

Reference Manual to LL API cross reference:

- 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 (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format

Reference Manual to LL API cross reference:

- 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 (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- 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 (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format

Reference Manual to LL API cross reference:

- 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 (BYPSHAD=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 (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- 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 WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmA Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- 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:

- ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- 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 WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmB Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- 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:

- ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set ALARM B time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  – LL_RTC_ALMB_TIME_FORMAT_AM
  – LL_RTC_ALMB_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get ALARM B time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_ALMB_TIME_FORMAT_AM
  – LL_RTC_ALMB_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- ALRMBR HT LL_RTC_ALMB_SetHour
- ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (RTC_TypeDef * RTCx)

Function description

Get ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- ALRMBR HT LL_RTC_ALMB_GetHour
- ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- ALRMBR MNT LL_RTC_ALMB_SetMinute
- ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (RTC_TypeDef * RTCx)

Function description

Get ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- 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:

- ALRMBR PM LL_RTC_ALMB_ConfigTime
- ALRMBR HT LL_RTC_ALMB_ConfigTime
- ALRMBR HU LL_RTC_ALMB_ConfigTime
- ALRMBR MNT LL_RTC_ALMB_ConfigTime
- ALRMBR MNU LL_RTC_ALMB_ConfigTime
- ALRMBR ST LL_RTC_ALMB_ConfigTime
- ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- ALRMBR HT LL_RTC_ALMB_GetTime
- ALRMBR HU LL_RTC_ALMB_GetTime
- ALRMBR MNT LL_RTC_ALMB_GetTime
- ALRMBR MNU LL_RTC_ALMB_GetTime
- ALRMBR ST LL_RTC_ALMB_GetTime
- ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** Value between Min_Data=0x00 and Max_Data=0xF

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- 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_PUSHPULL

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 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration output

LL_RTC_CALIB_OUTPUT_NONE

Calibration output disabled

LL_RTC_CALIB_OUTPUT_1HZ

Calibration output is 1 Hz

LL_RTC_CALIB_OUTPUT_512HZ

Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC

Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_16SEC

Use a 16-second calibration cycle period

LL_RTC_CALIB_PERIOD_8SEC

Use a 8-second calibration cycle period

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

Thrusday

LL_RTC_WEEKDAY_FRIDAY

Friday

LL_RTC_WEEKDAY_SATURDAY

Saturday

LL_RTC_WEEKDAY_SUNDAY

Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

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:

- RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)

Function description

Get Tx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_SetNSSMode

Function name

__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)

Function description

Set NSS mode.

Parameters

- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  – LL_SPI_NSS_SOFT
  – LL_SPI_NSS_HARD_INPUT
  – LL_SPI_NSS_HARD_OUTPUT

Return values

- **None:**

Notes

- LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 SSM LL_SPI_SetNSSMode
-
- CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)

Function description

Get NSS mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:

- CR1 SSM LL_SPI_GetNSSMode
-
- CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_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_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read 8-Bits in the data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read 16-Bits in the data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- DR DR LL_SPI_ReceiveData16

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write 8-Bits in the data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write 16-Bits in the data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_SPI_TransmitData16

LL_SPI_DeInit

Function name

ErrorStatus LL_SPI_DeInit (SPI_TypeDef * SPIx)

Function description

De-initialize the SPI registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: SPI registers are de-initialized
  – ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name

ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Initialize the SPI registers according to the specified parameters in SPI_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name

void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_Enable

Function name

__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)

Function description

Select I2S mode and Enable I2S peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SMOD LL_I2S_Enable
- I2SCFGR I2SE LL_I2S_Enable

LL_I2S_Disable

Function name

__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)

Function description

Disable I2S peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SE LL_I2S_Disable

LL_I2S_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabled (SPI_TypeDef * SPIx)

Function description

Check if I2S peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SCFGR I2SE LL_I2S_IsEnabled

LL_I2S_SetDataFormat

Function name

__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataFormat)

Function description

Set I2S data frame length.

Parameters

- **SPIx:** SPI Instance
- **DataFormat:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name

__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (SPI_TypeDef * SPIx)

Function description

Get I2S data frame length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_GetDataFormat
- I2SCFGR CHLEN LL_I2S_GetDataFormat

LL_I2S_SetClockPolarity

Function name

__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set I2S clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (SPI_TypeDef * SPIx)

Function description

Get I2S clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_I2S_POLARITY_LOW
  – LL_I2S_POLARITY_HIGH

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name

__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S standard protocol.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  – LL_I2S_STANDARD_PHILIPS
  – LL_I2S_STANDARD_MSB
  – LL_I2S_STANDARD_LSB
  – LL_I2S_STANDARD_PCM_SHORT
  – LL_I2S_STANDARD_PCM_LONG

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name

__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)

Function description

Get I2S standard protocol.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name

__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set I2S transfer mode.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (SPI_TypeDef * SPIx)

Function description

Get I2S transfer mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_SetPrescalerLinear

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint8_t PrescalerLinear)

Function description

Set I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (SPI_TypeDef * SPIx)

Function description

Get I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- I2SPR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)

Function description

Set I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  – LL_I2S_PRESCALER_PARITY_EVEN
  – LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (SPI_TypeDef * SPIx)

Function description

Get I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_I2S_PRESCALER_PARITY_EVEN
  – LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SPR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name

__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)

Function description

Enable the master clock ouput (Pin MCK)

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name

__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)

Function description

Disable the master clock ouput (Pin MCK)

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (SPI_TypeDef * SPIx)

Function description

Check if the master clock ouput (Pin MCK) is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_EnableAsyncStart

Function name

__STATIC_INLINE void LL_I2S_EnableAsyncStart (SPI_TypeDef * SPIx)

Function description

Enable asynchronous start.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_EnableAsyncStart

LL_I2S_DisableAsyncStart

Function name

__STATIC_INLINE void LL_I2S_DisableAsyncStart (SPI_TypeDef * SPIx)

Function description

Disable asynchronous start.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_DisableAsyncStart

LL_I2S_IsEnabledAsyncStart

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledAsyncStart (SPI_TypeDef * SPIx)

Function description

Check if asynchronous start is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_IsEnabledAsyncStart

LL_I2S_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE (SPI_TypeDef * SPIx)

Function description

Check if Rx buffer is not empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_I2S_IsActiveFlag_RXNE

LL_I2S_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE (SPI_TypeDef * SPIx)

Function description

Check if Tx buffer is empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXE LL_I2S_IsActiveFlag_TXE

LL_I2S_IsActiveFlag_BSY

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY (SPI_TypeDef * SPIx)

Function description

Get busy flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSY LL_I2S_IsActiveFlag_BSY

LL_I2S_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (SPI_TypeDef * SPIx)

Function description

Get overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (SPI_TypeDef * SPIx)

Function description

Get underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (SPI_TypeDef * SPIx)

Function description

Get frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR FRE LL_I2S_IsActiveFlag_FRE

LL_I2S_IsActiveFlag_CHSIDE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_CHSIDE (SPI_TypeDef * SPIx)

Function description

Get channel side flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- 0: Channel Left has to be transmitted or has been received 1: Channel Right has to be transmitted or has been received It has no significance in PCM mode.

Reference Manual to LL API cross reference:

- SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE

LL_I2S_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR FRE LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)

Function description

Enable error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_EnableIT_ERR

LL_I2S_EnableIT_RXNE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Enable Rx buffer not empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_EnableIT_RXNE

LL_I2S_EnableIT_TXE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_TXE (SPI_TypeDef * SPIx)

Function description

Enable Tx buffer empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_EnableIT_TXE

LL_I2S_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_ERR (SPI_TypeDef * SPIx)

Function description

Disable error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_DisableIT_ERR

LL_I2S_DisableIT_RXNE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Disable Rx buffer not empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_DisableIT_RXNE

LL_I2S_DisableIT_TXE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_TXE (SPI_TypeDef * SPIx)

Function description

Disable Tx buffer empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_DisableIT_TXE

LL_I2S_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_ERR (SPI_TypeDef * SPIx)

Function description

Check if ERR IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_IsEnabledIT_ERR

LL_I2S_IsEnabledIT_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXNE (SPI_TypeDef * SPIx)

Function description

Check if RXNE IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_IsEnabledIT_RXNE

LL_I2S_IsEnabledIT_TXE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXE (SPI_TypeDef * SPIx)

Function description

Check if TXE IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_IsEnabledIT_TXE

LL_I2S_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read 16-Bits in data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- DR DR LL_I2S_ReceiveData16

LL_I2S_TransmitData16

Function name

__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write 16-Bits in data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_I2S_TransmitData16

LL_I2S_DeInit

Function name

ErrorStatus LL_I2S_DeInit (SPI_TypeDef * SPIx)

Function description

De-initialize the SPI/I2S registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_I2S_Init

Function name

ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_I2S_StructInit

Function name

void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Set each LL_I2S_InitTypeDef field to default value.

Parameters

- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_ConfigPrescaler

Function name

void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)

Function description

Set linear and parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** value Min_Data=0x02 and Max_Data=0xFF.
- **PrescalerParity:** This parameter can be one of the following values:
  – LL_I2S_PRESCALER_PARITY_EVEN
  – LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Notes

- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).

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=0xFFF

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_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_SetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_SetTracePinAssignment

LL_DBGMCU_GetTracePinAssignment

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetTracePinAssignment (void)

Function description

Get Trace pin assignment control.

Return values

- **Returned:** value can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_GetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_GetTracePinAssignment

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP (*)
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP (*)
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP (*)
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP (*)
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP (*)
  - LL_DBGMCU_APB1_GRP1_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 RMII 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 OSSIState***
- ***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::OSSIState***
  Specifies the Off-State used in Idle state. This parameter can be a value of *TIM_LL_EC_OSSI*This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***
  Specifies the LOCK level parameters. This parameter can be a value of *TIM_LL_EC_LOCKLEVEL*
  Note:
  – The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.
- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***
  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.

- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***
  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of *TIM_LL_EC_BREAK_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***
  Specifies the TIM Break Input pin polarity. This parameter can be a value of *TIM_LL_EC_BREAK_POLARITY*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::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_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2

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_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2

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_OCPOLARITY_HIGH
  – LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity

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_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity

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_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 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 connectected to the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.

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_EnableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)

Function description

Generate an update event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 1 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 3 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 4 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)

Function description

Generate commutation event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)

Function description

Generate trigger event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)

Function description

Generate break event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_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 1 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_9TRANSFERS

Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS

Transfer is done to 10 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_11TRANSFERS

Transfer is done to 11 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_12TRANSFERS

Transfer is done to 12 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_13TRANSFERS

Transfer is done to 13 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_14TRANSFERS

Transfer is done to 14 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_15TRANSFERS

Transfer is done to 15 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_16TRANSFERS

Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS

Transfer is done to 17 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_18TRANSFERS

Transfer is done to 18 registers starting from the DMA burst base address

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 TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING

The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted

LL_TIM_IC_POLARITY_BOTHEDGE

The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

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, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

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 (TI12P2) 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_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name

__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)

Function description

Select the phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPhase (USART_TypeDef * USARTx)

Function description

Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name

__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description

Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (USART_TypeDef * USARTx)

Function description

Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name

__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description

Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters

- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  – LL_USART_PHASE_1EDGE
  – LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  – LL_USART_POLARITY_LOW
  – LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  – LL_USART_LASTCLKPULSE_NO_OUTPUT
  – LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_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 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (USART_TypeDef * USARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_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() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode
- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name

__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in LIN Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.
- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function
- Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_ConfigLINMode
- CR2 STOP LL_USART_ConfigLINMode
- CR2 LINEN LL_USART_ConfigLINMode
- CR3 IREN LL_USART_ConfigLINMode
- CR3 SCEN LL_USART_ConfigLINMode
- CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name

__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Half Duplex Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name

__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name

__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- 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_VALUEThis 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 WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Prescaler Value.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name

__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description

Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters

- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values

- **None:**

Notes

- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Window Value (7 bits value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Enable the Early Wakeup Interrupt.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Check if Early Wakeup Interrupt is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

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 function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in 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 systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).

2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() 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	Added CEC, FMPI2C, QSPI and SPDIFRX in Table 1. Added STM32F446xx, cec, dcmi, fmpi2c, fmpi2c_ex, spdifrx and qspi in Table 1. Updated Common macros section in HAL common resources. 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.
15-Sep-2015	3	Added DSI and LPTIM and removed msp_template in Table 1. Added STM32F469xx, STM32F479xx, STM32F410xx, dsi, ltdc_ex and lptim in Table 1. Added HAL DSI Generic Driver, HAL LPTIM Generic Driver and HAL LTDC Generic Driver.
02-Sep-2016	4	Added DFSDM in Acronyms and definitions. Added STM32F412Cx, STM32F412Rx, STM32F412Vx, STM32F412Zx and DFSDM in Table 1. Added HAL DFSDM Generic Driver.
20-Feb-2017	5	Added MMC in Section 2 Acronyms and definitions. Added STM32F413xx, STM32F423xx, MMC row and LL driver rows in Table : "List of devices supported by HAL drivers". Added Section 43: "HAL MMC Generic Driver" description. Added description of LL Generic drivers. Updated FAQ section.
31-Jul-2020	6	Updated Section Introduction. Added Section 1 General information. Updated Section 2 Acronyms and definitions. Updated Figure 1. Example of project template. Added Section 4 Overview of low-layer drivers and Section 5 Cohabiting of HAL and LL. Updated HAL drivers. Added low-layer drivers.
18-Jun-2021	7	Section 40 HAL IRDA Generic Driver: updated note at the end of Section 40.2.2 Callback registration. Section 70 HAL UART Generic Driver: • Added support for the following new APIs: HAL_UARTEx_ReceiveToIdle(), HAL_UARTEx_ReceiveToIdle_IT() and HAL_UARTEx_ReceiveToIdle_DMA(). • Updated note at the end of Section 70.2.2 Callback registration. Section 71 HAL USART Generic Driver: updated note at the end of Section 71.2.2 Callback registration.

Contents

List of tables

List of figures