UM1725
User Manual

Description of STM32F4xx HAL drivers

Introduction

STM32Cube™ is an STMicroelectronics original initiative to ease developers life by reducing development efforts, time and cost. STM32Cube™ covers STM32 portfolio.

STM32Cube™ Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for STM32F4 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

The HAL drivers layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees an easy portability on other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

The HAL drivers 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 family or part number.

The HAL drivers are feature-oriented instead of IP-oriented. As an example, the timer APIs are split into several categories following the functions offered by the IP: basic timer, capture, pulse width modulation (PWM), etc..

The drivers source code is developed in Strict ANSI-C which makes it independent from the development tools. It is checked with CodeSonar™ static analysis tool. It is fully documented and is MISRA-C 2004 compliant.

The HAL drivers layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking contributes to enhance the firmware robustness. Run-time detection is also suitable for user application development and debugging.

This user manual is structured as follows:

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

Contents

List of tables

List of figures

1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DCMI	Digital Camera Module Interface
DMA	Direct Memory Access
DMA2D	Chrom-Art Accelerator™ controller
ETH	Ethernet controller
EXTI	External interrupt/event controller
FLASH	Flash memory
FSMC	Flexible Static Memory controller
FMC	Flexible Memory controller
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB Host Controller Driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LTDC	LCD TFT Display Controller
MSP	MCU Specific Package
NAND	NAND external Flash memory
NOR	NOR external Flash memory
NVIC	Nested Vectored Interrupt Controller
PCCARD	PCCARD external memory
PCD	USB Peripheral Controller Driver

Acronym	Definition
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SAI	Serial Audio Interface
SD	Secure Digital
SDRAM	SDRAM external memory
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

2 Overview of HAL drivers

The HAL drivers were 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 consist of a set of driver modules, each module being linked to 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 folllowing:

- 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.
- Peripheral multi-instance support allowing concurrent API calls for multiple instances of a given peripheral (USART1, 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.

2.1 HAL and user application files

2.1.1 HAL driver files

A HAL drivers are composed of the following set of files:

Table 2: HAL drivers 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, …*

File	Description
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_dma_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_dma_ex.h, …*
stm32f4xx_ll_ppp.c	Peripheral low layer driver that can be accessed from one or more HAL drivers. It offers a set of APIs and services used by the upper driver. From the user point of view, low-level drivers are not accessible directly. They are used only by the HAL drivers built upon them. *Example: stm32f4xx_ll_fmc.c offers a set of API used by stm32f4xx_hal_sdram.c, stm32f4xx_hal_sram.c, stm32f4xx_hal_nor.c, stm32f4xx_hal_nand.c,…*
stm32f4xx_ll_ppp.h	Header file of the low layer C file. It is included in the HAL driver header file, thus making the low-level driver an intrinsic add-on of the HAL driver that is not visible from the application. *Example: stm32f4xx_ll_fmc.h, stm32f4xx_ll_usb.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.

Since the low level drivers are only used by the HAL drivers built upon them, the APIs provided by these drivers will not be described in this document.

2.1.2 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() which 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 to : • relocate the vector table in internal SRAM. • configure FSMC/FMC peripheral (when available) to use as data memory the external SRAM or SDRAM mounted on the evaluation board.
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 to adapt 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.
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 stm32*f4xx*_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: • the call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

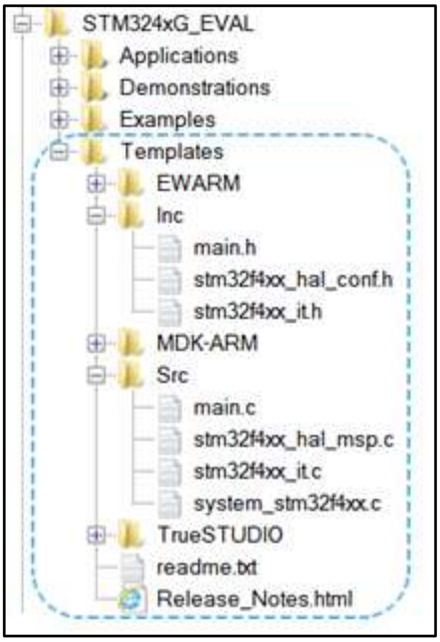
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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

- It contains 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 to configure 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 frequency of the device

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

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

2.2.1 Peripheral handle structures

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

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;
```

1) The multi-instance feature implies that all the APIs used in the application are re-entrant and avoid using global variables because a subroutine can fail to be re-entrant 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:

- Re-entrant code does not hold any static (or global) non-constant data: re-entrant functions can work with global data. For example, a re-entrant interrupt service routine can grab a piece of hardware status to work with (e.g. 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.

2.2.2 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;
```

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)
```

2.2.3 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)
```

2.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL drivers 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 || */
```

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: APIs classification

	Generic file	Extension file
Common APIs	X	X [(1)]
Family specific APIs		X
Device specific APIs		X

Notes:

(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

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.

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

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	STM32F 405xx	STM32F 415xx	STM32F 407xx	STM32F 417xx	STM32F 427xx	STM32F 437xx	STM32F 429xx	STM32F 439xx	STM32F 401xC	STM32F 401xE
stm32f4xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_ adc.c	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
stm32f4xx_hal_ can.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_ cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_ crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_ cryp.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No
stm32f4xx_hal_ cryp_ex.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No
stm32f4xx_hal_ dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_ dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No

IP/Module	STM32F 405xx	STM32F 415xx	STM32F 407xx	STM32F 417xx	STM32F 427xx	STM32F 437xx	STM32F 429xx	STM32F 439xx	STM32F 401xC	STM32F 401xE
stm32f4xx_hal_dcmi.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_dma2d.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_eth.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_flash.c	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
stm32f4xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_hash.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No
stm32f4xx_hal_hash_ex.c	No	No	No	No	No	Yes	No	Yes	No	No
stm32f4xx_hal_hcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_i2c.c	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
stm32f4xx_hal_i2s.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_ltdc.c	No	No	No	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_nand.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_nor.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_pccard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_pcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32F 405xx	STM32F 415xx	STM32F 407xx	STM32F 417xx	STM32F 427xx	STM32F 437xx	STM32F 429xx	STM32F 439xx	STM32F 401xC	STM32F 401xE
stm32f4xx_hal_pwr_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_rcc.c	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
stm32f4xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_rtc.c	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
stm32f4xx_hal_sai.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_sd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_sdram.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_sram.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_tim.c	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
stm32f4xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_ll_fmc.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No
stm32f4xx_ll_fsmc.c	Yes	Yes	Yes	Yes	No	No	No	No	No	No
stm32f4xx_ll_sdmmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_ll_usb.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.5 HAL drivers rules

2.5.1 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_StructnameType Def*	*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 STM32F4xx reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in stm32f4xxx.h header file. stm32f4xxx.h corresponds to stm32f401xc.h, stm32f401xe.h, stm32f405xx.h, stm32f415xx.h, stm32f407xx.h, stm32f417xx.h, stm32f427xx.h, stm32f437xx.h, stm32f429xx.h or stm32f439xx.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 (e.g. 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 (e.g. ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (e.g. 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 (e.g. HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named PPP_DeInit, e.g. 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_ADC_InjectionStart*() refers to the injection mode

2.5.2 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: 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 = ".

- 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)
```

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

2.6 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 (PWM, OC, 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 runtime 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

Function Group	Common API Name	Description
	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 runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 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
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)

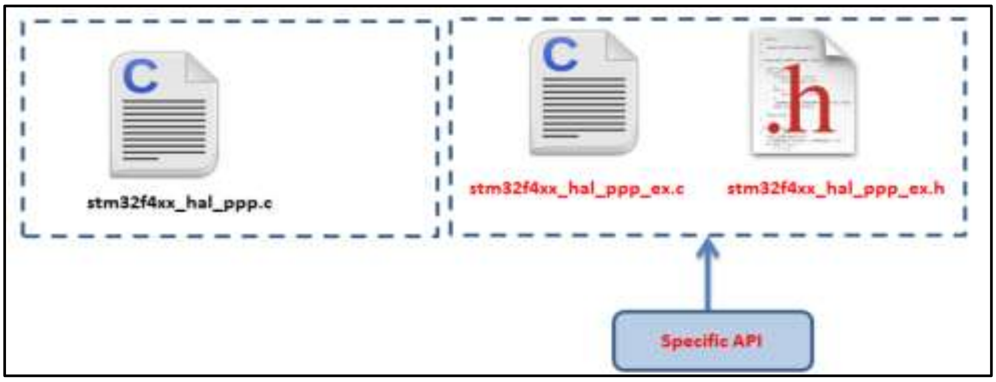
2.7.2 HAL extension model cases

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

Case1: 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



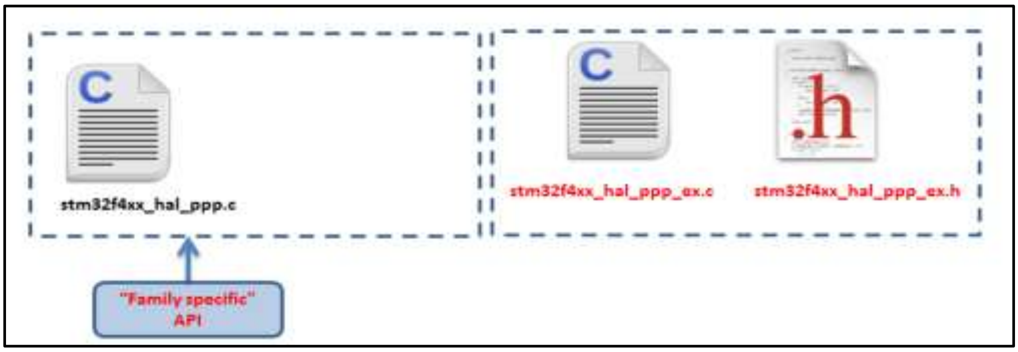
Example: stm32f4xx_hal_flash_ex.c/h

```
#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 || */
```

Case2: 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

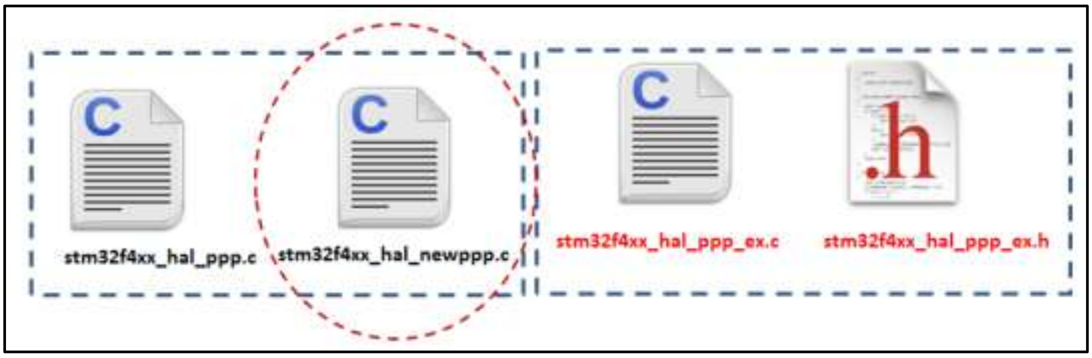
```
HAL_StatusTypeDef HAL_ADCEx_InjectedStop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

Case3 : 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 are added in stm32f4xx_hal_newppp.c. However the inclusion of this file is selected in the stm32fxx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

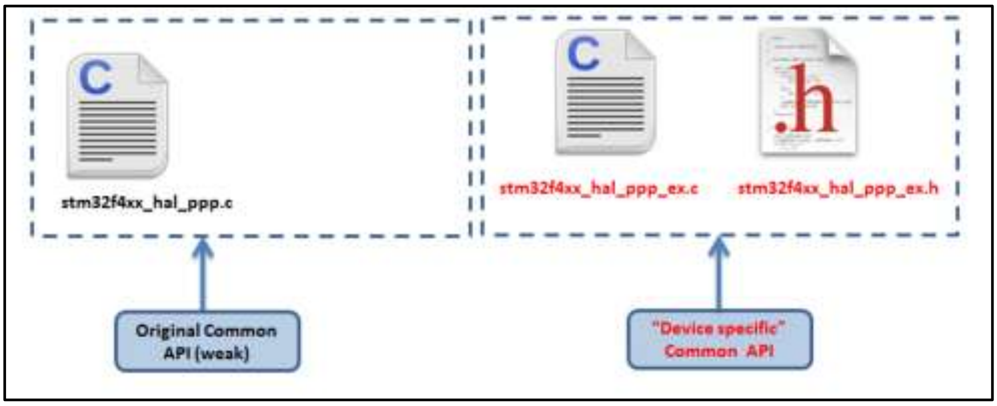


Example: stm32f4xx_hal_sai.c/h

Case4: 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 will overwrite the original routine by the new defined function.

Figure 5: Updating existing APIs



Case5 : Updating existing data structures

The data structure for a specific device part number (e.g. 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.

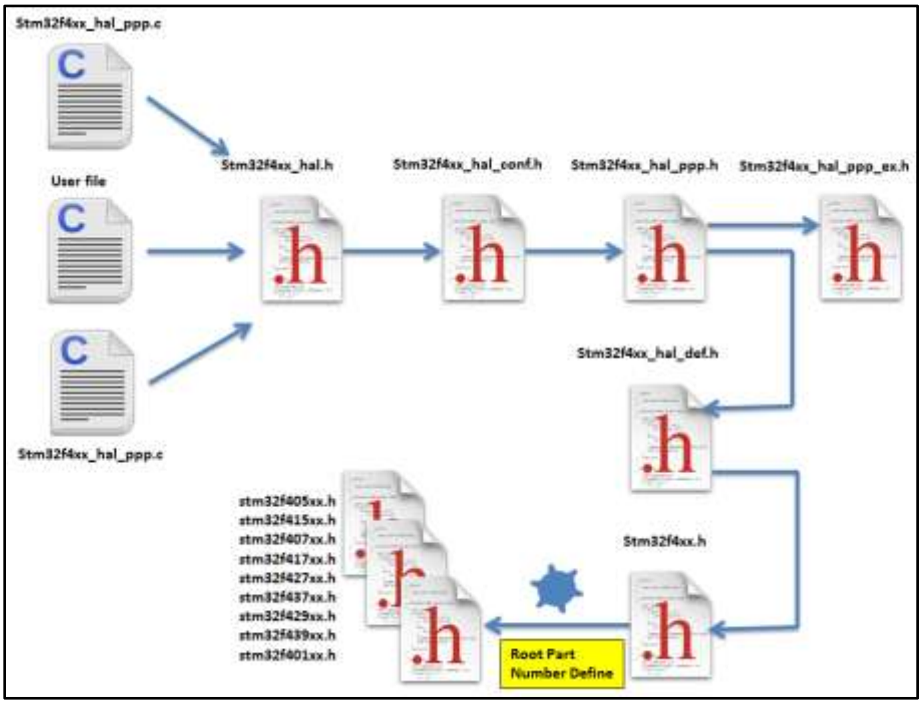
Example:

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

2.8 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
(…)
```

2.9 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 (Write register, Read register…etc.).
- **Common macros**
  – Macros defining NULL and HAL_MAX_DELAY

```
#ifndef NULL
#define NULL (void *) 0
#endif
#define HAL_MAX_DELAY 0xFFFFFFFF
```

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

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

    } while(0)
```

2.10 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)
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)

Configuration item	Description	Default Value
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 receive	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	

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.

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.

2.11 HAL system peripheral handling

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

2.11.1 Clock

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

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).

- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - Selects the system clock source
  - Configures AHB, APB1 and APB2 clock dividers
  - Configures the number od Flash memory wait states
  - Updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, SDIO, I2S, SAI, Audio PLL…). In this case, the clock configuration is performed by an extended API defined in stm32f4xx_hal_ppp_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-init function that return clock configuration to reset state
- Get clock functions that allow retreiving various clock configurations (system clock, HCLK, PCLK1, PCLK2, …)
- MCO and CSS configuration functions

A set of macros are defined in stm32f4xx_hal_rcc.h. They allows executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __PPP_CLK_ENABLE/__PPP_CLK_DISABLE to enable/disable the peripheral clock
- __PPP_FORCE_RESET/__PPP_RELEASE_RESET to force/release peripheral reset
- __PPP_CLK_SLEEP_ENABLE/__PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during low power (Sleep) mode.

2.11.2 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), 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]

Structure field	Description
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 GPIO_SPEED_HIGH
Alternate	Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PPP, where AFx: is the alternate function index PPP: is the peripheral instance Example: use GPIO_AF1_TIM1 to connect TIM1 IOs on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines. 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

```
LEDsGPIO_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);
```

2.11.3 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()

2.11.4 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_PVDConfig()
  - 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, for STM32F429/F439xx only
  - HAL_PWREx_ActivateOverDrive()
  - HAL_PWREx_EnableFlashPowerDown().

2.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral. As a result there are no EXTI APIs but each peripheral HAL driver implements the associated EXTI configuration and EXTI function are implemented as macros in its header file.

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

The EXTI lines connected internally to the PVD, RTC, USB, and COMP 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().

2.11.6 DMA

The DMA HAL driver allows enabling and configuring the peripheral to be connected to the DMA Stream (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 stream, 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
- Stream 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
  a. 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.
  b. 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
  a. Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
  b. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
  c. 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.
  d. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
  e. 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: enablse the specified DMA Stream.
- __HAL_DMA_DISABLE: disables the specified DMA Stream.
- __HAL_DMA_GET_FS: returns the current DMA Stream FIFO filled level.
- __HAL_DMA_GET_FLAG: gets the DMA Stream pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA Stream pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA Stream interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA Stream interrupts.
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA stream interrupt has occurred or not.

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

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

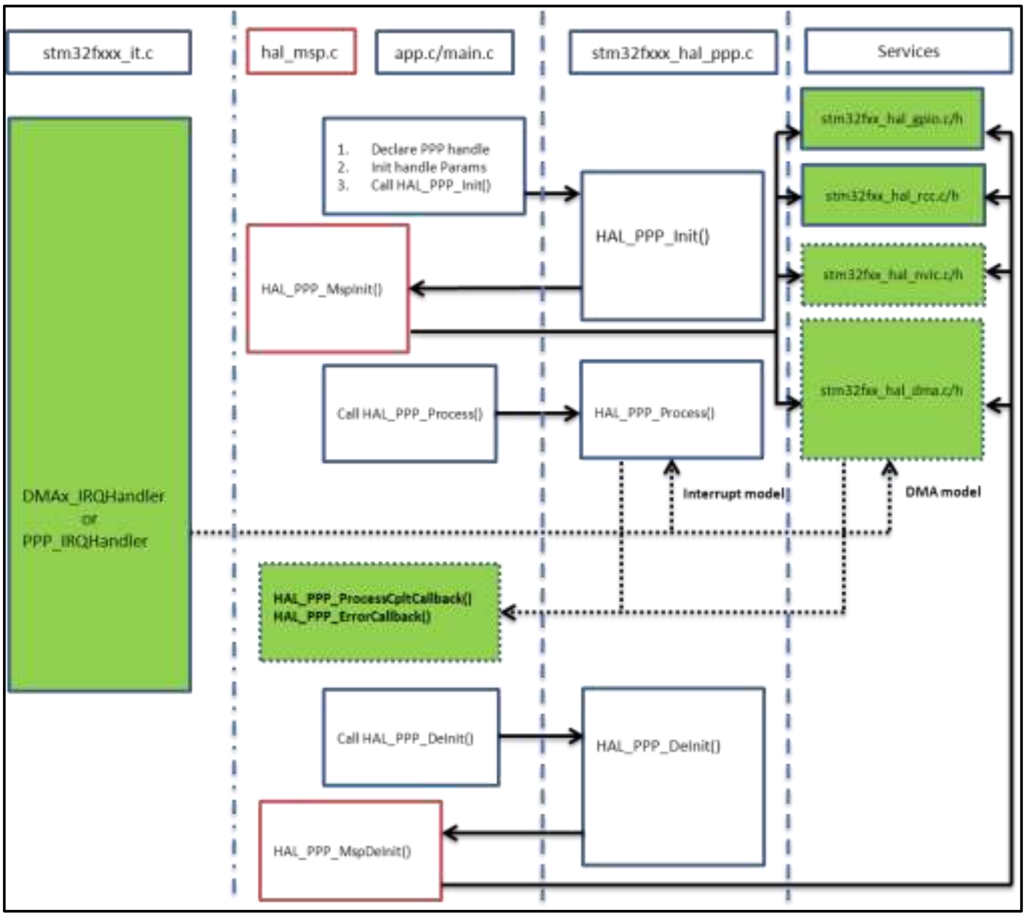
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

2.12 How to use HAL drivers

2.12.1 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



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.

2.12.2 HAL initialization

2.12.2.1 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 a is 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 will be blocked.

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.

2.12.2.2 HAL 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); }
```

2.12.2.3 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

Routine	Description
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.

2.12.3 HAL IO 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

2.12.3.1 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 HELIAC; }
```

2.12.3.2 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()*: launch the process
- *HAL_PPP_IRQHandler()*: the global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: the 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);
 }
```

2.12.3.3 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 ;
(…)
}
```

2.12.4 Timeout and error management

2.12.4.1 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 value 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 (1)	Infinite poll till process is successful

Notes:

(1)HAL_MAX_DELAY is defined in the stm32fxxx_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;
 }
 }
 (…)
}
```

2.12.4.2 Error management

The HAL drivers implement a check for the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system can crash or go into an undefined state. These critical parameters are checked before they are 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 */
}
```

2.12.4.3 Run-time checking

The HAL implements run-time failure detection by checking the input values of all HAL drivers functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL drivers' 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 *stm32f34xx_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)
{
 }
}
```

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.

3 HAL common driver

3.1 HAL Firmware driver API description

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

3.1.1 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

3.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize 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-Initialize common part of the HAL
- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***

3.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- 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
- ***HAL_IncTick()***
- ***HAL_GetTick()***
- ***HAL_Delay()***
- ***HAL_GetHalVersion()***
- ***HAL_GetREVID()***
- ***HAL_GetDEVID()***
- ***HAL_EnableDBGSleepMode()***
- ***HAL_DisableDBGSleepMode()***
- ***HAL_EnableDBGStopMode()***

- *HAL_DisableDBGStopMode()*
- *HAL_EnableDBGStandbyMode()*
- *HAL_DisableDBGStandbyMode()*
- *HAL_EnableCompensationCell()*
- *HAL_DisableCompensationCell()*
- *HAL_EnableMemorySwappingBank()*
- *HAL_DisableMemorySwappingBank()*

3.1.4 Initialization and de-initialization Functions

3.1.4.1 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.
Parameters	• None.
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.

3.1.4.2 HAL_DeInit

Function Name	**HAL_StatusTypeDef HAL_DeInit (void)**
Function Description	This function de-Initializes common part of the HAL and stops the systick.
Parameters	• None.
Return values	• **HAL status**
Notes	• None.

3.1.4.3 HAL_MspInit

Function Name	**void HAL_MspInit (void)**
Function Description	Initializes the MSP.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.4.4 HAL_MspDeInit

Function Name	**void HAL_MspDeInit (void)**
Function Description	DeInitializes the MSP.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5 HAL Control functions

3.1.5.1 HAL_IncTick

Function Name	**void HAL_IncTick (void)**
Function Description	This function is called from SysTick ISR each 1 millisecond, to increment a global variable "uwTick" used as time base.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.2 HAL_GetTick

Function Name	**uint32_t HAL_GetTick (void)**
Function Description	Povides a tick value in millisecond.
Parameters	• None.
Return values	• **tick value**
Notes	• None.

3.1.5.3 HAL_Delay

Function Name	**void HAL_Delay (__IO uint32_t Delay)**
Function Description	Provides a blocking delay in millisecond.
Parameters	• **Delay :** specifies the delay time length, in milliseconds.
Return values	• None.
Notes	• Care must be taken when using HAL_Delay(), this function provides accurate delay (in milliseconds) based on 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. To change the SysTick interrupt priority you have to use HAL_NVIC_SetPriority() function.

3.1.5.4 HAL_GetHalVersion

Function Name	**uint32_t HAL_GetHalVersion (void)**
Function Description	Returns the HAL revision.
Parameters	• None.
Return values	• **version : 0xXYZR (8bits for each decimal, R for RC)**
Notes	• None.

3.1.5.5 HAL_GetREVID

Function Name	**uint32_t HAL_GetREVID (void)**
Function Description	Returns the device revision identifier.
Parameters	• None.
Return values	• **Device revision identifier**
Notes	• None.

3.1.5.6 HAL_GetDEVID

Function Name	**uint32_t HAL_GetDEVID (void)**
Function Description	Returns the device identifier.
Parameters	• None.
Return values	• **Device identifier**
Notes	• None.

3.1.5.7 HAL_EnableDBGSleepMode

Function Name	**void HAL_EnableDBGSleepMode (void)**
Function Description	Enable the Debug Module during SLEEP mode.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.8 HAL_DisableDBGSleepMode

Function Name	**void HAL_DisableDBGSleepMode (void)**
Function Description	Disable the Debug Module during SLEEP mode.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.9 HAL_EnableDBGStopMode

Function Name	**void HAL_EnableDBGStopMode (void)**
Function Description	Enable the Debug Module during STOP mode.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.10 HAL_DisableDBGStopMode

Function Name	**void HAL_DisableDBGStopMode (void)**
Function Description	Disable the Debug Module during STOP mode.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.11 HAL_EnableDBGStandbyMode

Function Name	**void HAL_EnableDBGStandbyMode (void)**

Function Description	Enable the Debug Module during STANDBY mode.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.12 HAL_DisableDBGStandbyMode

Function Name	**void HAL_DisableDBGStandbyMode (void)**
Function Description	Disable the Debug Module during STANDBY mode.
Parameters	• None.
Return values	• None.
Notes	• None.

3.1.5.13 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.

3.1.5.14 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.

3.1.5.15 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 devices. • Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08100000 (and aliased at 0x00100000)

3.1.5.16 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 devices. • The default state : Flash Bank1 mapped at 0x08000000 (and aliased @0x0000 0000) and Flash Bank2 mapped at 0x08100000 (and aliased at 0x00100000)

3.2 HAL Firmware driver defines

3.2.1 HAL

HAL

4 HAL ADC Generic Driver

4.1 ADC Firmware driver registers structures

4.1.1 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 HAL_ADC_StateTypeDef 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 HAL_ADC_StateTypeDef ADC_HandleTypeDef::State***
  - ADC communication state
- ***__IO uint32_t ADC_HandleTypeDef::ErrorCode***
  - ADC Error code

4.1.2 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***
- ***uint32_t ContinuousConvMode***

- ***uint32_t DMAContinuousRequests***
- ***uint32_t NbrOfConversion***
- ***uint32_t DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t ExternalTrigConv***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***
  - Select the frequency of the clock to the ADC. 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 dual mode. This parameter can be a value of ***ADC_Resolution***
- ***uint32_t ADC_InitTypeDef::DataAlign***
  - Specifies whether the ADC data alignment is left or right. This parameter can be a value of ***ADC_data_align***
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  - Specifies whether the conversion is performed in Scan (multi channels) or Single (one channel) mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_InitTypeDef::EOCSelection***
  - Specifies whether the EOC flag is set at the end of single channel conversion or at the end of all conversions. This parameter can be a value of ***ADC_EOCSelection***
- ***uint32_t ADC_InitTypeDef::ContinuousConvMode***
  - Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::DMAContinuousRequests***
  - Specifies whether the DMA requests is performed in Continuous or in Single mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
  - Specifies the number of ADC conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min_Data = 1 and Max_Data = 16.
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode***
  - Specifies whether the conversion is performed in Discontinuous or not for regular channels. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
  - Specifies the number of ADC discontinuous conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
  - Select the external trigger edge and enable the trigger of a regular group. This parameter can be a value of ***ADC_External_trigger_edge_Regular***
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  - Select the external event used to trigger the start of conversion of a regular group. This parameter can be a value of ***ADC_External_trigger_Source_Regular***

4.1.3 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***
  – The ADC channel to configure. This parameter can be a value of ***ADC_channels***
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  – 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***
  – The sample time value to be set for the selected channel. This parameter can be a value of ***ADC_sampling_times***
- ***uint32_t ADC_ChannelConfTypeDef::Offset***
  – Reserved for future use, can be set to 0

4.1.4 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***
- ***uint32_t 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.***
- ***uint32_t 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

4.1.5 ADC_Common_TypeDef

ADC_Common_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CSR***
- ***__IO uint32_t CCR***
- ***__IO uint32_t CDR***

Field Documentation

- ***__IO uint32_t ADC_Common_TypeDef::CSR***
  - ADC Common status register, Address offset: ADC1 base address + 0x300
- ***__IO uint32_t ADC_Common_TypeDef::CCR***
  - ADC common control register, Address offset: ADC1 base address + 0x304
- ***__IO uint32_t ADC_Common_TypeDef::CDR***
  - ADC common regular data register for dual AND triple modes, Address offset: ADC1 base address + 0x308

4.1.6 ADC_TypeDef

ADC_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t SR***
- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t SMPR1***
- ***__IO uint32_t SMPR2***
- ***__IO uint32_t JOFR1***
- ***__IO uint32_t JOFR2***
- ***__IO uint32_t JOFR3***
- ***__IO uint32_t JOFR4***
- ***__IO uint32_t HTR***
- ***__IO uint32_t LTR***

- ***__IO uint32_t SQR1***
- ***__IO uint32_t SQR2***
- ***__IO uint32_t SQR3***
- ***__IO uint32_t JSQR***
- ***__IO uint32_t JDR1***
- ***__IO uint32_t JDR2***
- ***__IO uint32_t JDR3***
- ***__IO uint32_t JDR4***
- ***__IO uint32_t DR***

Field Documentation

- ***__IO uint32_t ADC_TypeDef::SR***
  – ADC status register, Address offset: 0x00
- ***__IO uint32_t ADC_TypeDef::CR1***
  – ADC control register 1, Address offset: 0x04
- ***__IO uint32_t ADC_TypeDef::CR2***
  – ADC control register 2, Address offset: 0x08
- ***__IO uint32_t ADC_TypeDef::SMPR1***
  – ADC sample time register 1, Address offset: 0x0C
- ***__IO uint32_t ADC_TypeDef::SMPR2***
  – ADC sample time register 2, Address offset: 0x10
- ***__IO uint32_t ADC_TypeDef::JOFR1***
  – ADC injected channel data offset register 1, Address offset: 0x14
- ***__IO uint32_t ADC_TypeDef::JOFR2***
  – ADC injected channel data offset register 2, Address offset: 0x18
- ***__IO uint32_t ADC_TypeDef::JOFR3***
  – ADC injected channel data offset register 3, Address offset: 0x1C
- ***__IO uint32_t ADC_TypeDef::JOFR4***
  – ADC injected channel data offset register 4, Address offset: 0x20
- ***__IO uint32_t ADC_TypeDef::HTR***
  – ADC watchdog higher threshold register, Address offset: 0x24
- ***__IO uint32_t ADC_TypeDef::LTR***
  – ADC watchdog lower threshold register, Address offset: 0x28
- ***__IO uint32_t ADC_TypeDef::SQR1***
  – ADC regular sequence register 1, Address offset: 0x2C
- ***__IO uint32_t ADC_TypeDef::SQR2***
  – ADC regular sequence register 2, Address offset: 0x30
- ***__IO uint32_t ADC_TypeDef::SQR3***
  – ADC regular sequence register 3, Address offset: 0x34
- ***__IO uint32_t ADC_TypeDef::JSQR***
  – ADC injected sequence register, Address offset: 0x38
- ***__IO uint32_t ADC_TypeDef::JDR1***
  – ADC injected data register 1, Address offset: 0x3C
- ***__IO uint32_t ADC_TypeDef::JDR2***
  – ADC injected data register 2, Address offset: 0x40
- ***__IO uint32_t ADC_TypeDef::JDR3***
  – ADC injected data register 3, Address offset: 0x44
- ***__IO uint32_t ADC_TypeDef::JDR4***
  – ADC injected data register 4, Address offset: 0x48
- ***__IO uint32_t ADC_TypeDef::DR***

– ADC regular data register, Address offset: 0x4C

4.2 ADC Firmware driver API description

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

4.2.1 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.

4.2.2 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 __ADC_CLK_ENABLE()
   b. ADC pins configuration
      – Enable the clock for the ADC GPIOs using the following function: __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 __DMAx_CLK_ENABLE()
      – Configure and enable two DMA streams stream for managing data transfer from peripheral to memory (output stream)
      – Associate the initilalized 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.

2. Configure the ADC Prescaler, conversion resolution and data alignment using the HAL_ADC_Init() function.
3. Configure the ADC regular 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_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
- __HAL_ADC_GET_RESOLUTION: Return resolution bits in CR1 register

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

4.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC.
- ***HAL_ADC_Init()***
- ***HAL_ADC_DeInit()***
- ***HAL_ADC_MspInit()***
- ***HAL_ADC_MspDeInit()***

4.2.4 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.
- ***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()***

4.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure regular channels.
- Configure injected channels.
- Configure multimode.
- Configure the analog watch dog.
- ***HAL_ADC_ConfigChannel()***

- ***HAL_ADC_AnalogWDGConfig()***

4.2.6 Peripheral State and errors functions

This subsection provides functions allowing to

- Check the ADC state
- Check the ADC Error
- ***HAL_ADC_GetState()***
- ***HAL_ADC_GetError()***

4.2.7 Initialization and de-initialization functions

4.2.7.1 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).

4.2.7.2 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**
Notes	• None.

4.2.7.3 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.
Notes	• None.

4.2.7.4 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.
Notes	• None.

4.2.8 IO operation functions

4.2.8.1 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**
Notes	•	None.

4.2.8.2 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.

4.2.8.3 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	• None.

4.2.8.4 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: – ***AWD_EVENT :*** ADC Analog watch Dog event. – ***OVR_EVENT :*** ADC Overrun event. • **Timeout :** Timeout value in millisecond.
Return values	• **HAL status**
Notes	• None.

4.2.8.5 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.**
Notes	• None.

4.2.8.6 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.

4.2.8.7 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.
Notes	• None.

4.2.8.8 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**
Notes	• None.

4.2.8.9 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**
Notes	• None.

4.2.8.10 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**
Notes	• None.

4.2.8.11 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.
Notes	• None.

4.2.8.12 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.
Notes	• None.

4.2.8.13 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.
Notes	• None.

4.2.8.14 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	• None.

4.2.9 Peripheral Control functions

4.2.9.1 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**
Notes	• None.

4.2.9.2 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	• None.

4.2.10 ADC Peripheral State functions

4.2.10.1 HAL_ADC_GetState

Function Name	**HAL_ADC_StateTypeDef 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**
Notes	• None.

4.2.10.2 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**
Notes	• None.

4.3 ADC Firmware driver defines

4.3.1 ADC

ADC

ADC_analog_watchdog_selection

- #define: ***ADC_ANALOGWATCHDOG_SINGLE_REG ((uint32_t)(ADC_CR1_AWDSGL | ADC_CR1_AWDEN))***
- #define: ***ADC_ANALOGWATCHDOG_SINGLE_INJEC ((uint32_t)(ADC_CR1_AWDSGL | ADC_CR1_JAWDEN))***
- #define: ***ADC_ANALOGWATCHDOG_SINGLE_REGINJEC ((uint32_t)(ADC_CR1_AWDSGL | ADC_CR1_AWDEN | ADC_CR1_JAWDEN))***
- #define: ***ADC_ANALOGWATCHDOG_ALL_REG ((uint32_t)ADC_CR1_AWDEN)***

- #define: ***ADC_ANALOGWATCHDOG_ALL_INJEC ((uint32_t)ADC_CR1_JAWDEN)***
- #define: ***ADC_ANALOGWATCHDOG_ALL_REGINJEC ((uint32_t)(ADC_CR1_AWDEN | ADC_CR1_JAWDEN))***
- #define: ***ADC_ANALOGWATCHDOG_NONE ((uint32_t)0x00000000)***

ADC_channels

- #define: ***ADC_CHANNEL_0 ((uint32_t)0x00000000)***
- #define: ***ADC_CHANNEL_1 ((uint32_t)ADC_CR1_AWDCH_0)***
- #define: ***ADC_CHANNEL_2 ((uint32_t)ADC_CR1_AWDCH_1)***
- #define: ***ADC_CHANNEL_3 ((uint32_t)(ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))***
- #define: ***ADC_CHANNEL_4 ((uint32_t)ADC_CR1_AWDCH_2)***
- #define: ***ADC_CHANNEL_5 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))***
- #define: ***ADC_CHANNEL_6 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))***
- #define: ***ADC_CHANNEL_7 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))***

- #define: ***ADC_CHANNEL_8 ((uint32_t)ADC_CR1_AWDCH_3)***
- #define: ***ADC_CHANNEL_9 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_0))***
- #define: ***ADC_CHANNEL_10 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1))***
- #define: ***ADC_CHANNEL_11 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))***
- #define: ***ADC_CHANNEL_12 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2))***
- #define: ***ADC_CHANNEL_13 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))***
- #define: ***ADC_CHANNEL_14 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))***
- #define: ***ADC_CHANNEL_15 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))***
- #define: ***ADC_CHANNEL_16 ((uint32_t)ADC_CR1_AWDCH_4)***
- #define: ***ADC_CHANNEL_17 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_0))***

- #define: ***ADC_CHANNEL_18 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_1))***

- #define: ***ADC_CHANNEL_TEMPSENSOR ((uint32_t)ADC_CHANNEL_16)***

- #define: ***ADC_CHANNEL_VREFINT ((uint32_t)ADC_CHANNEL_17)***

- #define: ***ADC_CHANNEL_VBAT ((uint32_t)ADC_CHANNEL_18)***

ADC_channels_type

- #define: ***ALL_CHANNELS ((uint32_t)0x00000001)***

- #define: ***REGULAR_CHANNELS ((uint32_t)0x00000002)***

reserved for future use

- #define: ***INJECTED_CHANNELS ((uint32_t)0x00000003)***

reserved for future use

ADC_ClockPrescaler

- #define: ***ADC_CLOCKPRESCALER_PCLK_DIV2 ((uint32_t)0x00000000)***

- #define: ***ADC_CLOCKPRESCALER_PCLK_DIV4 ((uint32_t)ADC_CCR_ADCPRE_0)***

- #define: ***ADC_CLOCKPRESCALER_PCLK_DIV6 ((uint32_t)ADC_CCR_ADCPRE_1)***

- #define: ***ADC_CLOCKPRESCALER_PCLK_DIV8 ((uint32_t)ADC_CCR_ADCPRE)***

ADC_data_align

- #define: ***ADC_DATAALIGN_RIGHT ((uint32_t)0x00000000)***

- #define: ***ADC_DATAALIGN_LEFT ((uint32_t)ADC_CR2_ALIGN)***

ADC_EOCSelection

- #define: ***EOC_SEQ_CONV ((uint32_t)0x00000000)***

- #define: ***EOC_SINGLE_CONV ((uint32_t)0x00000001)***

- #define: ***EOC_SINGLE_SEQ_CONV ((uint32_t)0x00000002)***

reserved for future use

ADC_Error_Code

- #define: ***HAL_ADC_ERROR_NONE ((uint32_t)0x00)***

No error

- #define: ***HAL_ADC_ERROR_OVR ((uint32_t)0x01)***

OVR error

- #define: ***HAL_ADC_ERROR_DMA ((uint32_t)0x02)***

DMA transfer error

ADC_Event_type

- #define: ***AWD_EVENT ((uint32_t)ADC_FLAG_AWD)***

- #define: ***OVR_EVENT ((uint32_t)ADC_FLAG_OVR)***

ADC_External_trigger_edge_Regular

- #define: ***ADC_EXTERNALTRIGCONVEDGE_NONE ((uint32_t)0x00000000)***
- #define: ***ADC_EXTERNALTRIGCONVEDGE_RISING ((uint32_t)ADC_CR2_EXTEN_0)***
- #define: ***ADC_EXTERNALTRIGCONVEDGE_FALLING ((uint32_t)ADC_CR2_EXTEN_1)***
- #define: ***ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING ((uint32_t)ADC_CR2_EXTEN)***

ADC_External_trigger_Source_Regular

- #define: ***ADC_EXTERNALTRIGCONV_T1_CC1 ((uint32_t)0x00000000)***
- #define: ***ADC_EXTERNALTRIGCONV_T1_CC2 ((uint32_t)ADC_CR2_EXTSEL_0)***
- #define: ***ADC_EXTERNALTRIGCONV_T1_CC3 ((uint32_t)ADC_CR2_EXTSEL_1)***
- #define: ***ADC_EXTERNALTRIGCONV_T2_CC2 ((uint32_t)(ADC_CR2_EXTSEL_1 | ADC_CR2_EXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGCONV_T2_CC3 ((uint32_t)ADC_CR2_EXTSEL_2)***
- #define: ***ADC_EXTERNALTRIGCONV_T2_CC4 ((uint32_t)(ADC_CR2_EXTSEL_2 | ADC_CR2_EXTSEL_0))***

- #define: ***ADC_EXTERNALTRIGCONV_T2_TRGO ((uint32_t)(ADC_CR2_EXTSEL_2 | ADC_CR2_EXTSEL_1))***
- #define: ***ADC_EXTERNALTRIGCONV_T3_CC1 ((uint32_t)(ADC_CR2_EXTSEL_2 | ADC_CR2_EXTSEL_1 | ADC_CR2_EXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGCONV_T3_TRGO ((uint32_t)ADC_CR2_EXTSEL_3)***
- #define: ***ADC_EXTERNALTRIGCONV_T4_CC4 ((uint32_t)(ADC_CR2_EXTSEL_3 | ADC_CR2_EXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGCONV_T5_CC1 ((uint32_t)(ADC_CR2_EXTSEL_3 | ADC_CR2_EXTSEL_1))***
- #define: ***ADC_EXTERNALTRIGCONV_T5_CC2 ((uint32_t)(ADC_CR2_EXTSEL_3 | ADC_CR2_EXTSEL_1 | ADC_CR2_EXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGCONV_T5_CC3 ((uint32_t)(ADC_CR2_EXTSEL_3 | ADC_CR2_EXTSEL_2))***
- #define: ***ADC_EXTERNALTRIGCONV_T8_CC1 ((uint32_t)(ADC_CR2_EXTSEL_3 | ADC_CR2_EXTSEL_2 | ADC_CR2_EXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGCONV_T8_TRGO ((uint32_t)(ADC_CR2_EXTSEL_3 | ADC_CR2_EXTSEL_2 | ADC_CR2_EXTSEL_1))***
- #define: ***ADC_EXTERNALTRIGCONV_Ext_IT11 ((uint32_t)ADC_CR2_EXTSEL)***

ADC_flags_definition

- #define: ***ADC_FLAG_AWD ((uint32_t)ADC_SR_AWD)***
- #define: ***ADC_FLAG_EOC ((uint32_t)ADC_SR_EOC)***
- #define: ***ADC_FLAG_JEOC ((uint32_t)ADC_SR_JEOC)***
- #define: ***ADC_FLAG_JSTRT ((uint32_t)ADC_SR_JSTRT)***
- #define: ***ADC_FLAG_STRT ((uint32_t)ADC_SR_STRT)***
- #define: ***ADC_FLAG_OVR ((uint32_t)ADC_SR_OVR)***

ADC_interrupts_definition

- #define: ***ADC_IT_EOC ((uint32_t)ADC_CR1_EOCIE)***
- #define: ***ADC_IT_AWD ((uint32_t)ADC_CR1_AWDIE)***
- #define: ***ADC_IT_JEOC ((uint32_t)ADC_CR1_JEOCIE)***
- #define: ***ADC_IT_OVR ((uint32_t)ADC_CR1_OVRIE)***

ADC_Resolution

- #define: ***ADC_RESOLUTION12b ((uint32_t)0x00000000)***

- #define: ***ADC_RESOLUTION10b ((uint32_t)ADC_CR1_RES_0)***
- #define: ***ADC_RESOLUTION8b ((uint32_t)ADC_CR1_RES_1)***
- #define: ***ADC_RESOLUTION6b ((uint32_t)ADC_CR1_RES)***

ADC_sampling_times

- #define: ***ADC_SAMPLETIME_3CYCLES ((uint32_t)0x00000000)***
- #define: ***ADC_SAMPLETIME_15CYCLES ((uint32_t)ADC_SMPR1_SMP10_0)***
- #define: ***ADC_SAMPLETIME_28CYCLES ((uint32_t)ADC_SMPR1_SMP10_1)***
- #define: ***ADC_SAMPLETIME_56CYCLES ((uint32_t)(ADC_SMPR1_SMP10_1 | ADC_SMPR1_SMP10_0))***
- #define: ***ADC_SAMPLETIME_84CYCLES ((uint32_t)ADC_SMPR1_SMP10_2)***
- #define: ***ADC_SAMPLETIME_112CYCLES ((uint32_t)(ADC_SMPR1_SMP10_2 | ADC_SMPR1_SMP10_0))***
- #define: ***ADC_SAMPLETIME_144CYCLES ((uint32_t)(ADC_SMPR1_SMP10_2 | ADC_SMPR1_SMP10_1))***
- #define: ***ADC_SAMPLETIME_480CYCLES ((uint32_t)ADC_SMPR1_SMP10)***

5 HAL ADC Extension Driver

5.1 ADCEx Firmware driver registers structures

5.1.1 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***
- ***uint32_t AutoInjectedConv***
- ***uint32_t InjectedDiscontinuousConvMode***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***uint32_t ExternalTrigInjecConv***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  – Configure the ADC injected channel. This parameter can be a value of ***ADC_channels***
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  – The rank in the injected group sequencer This parameter must be a number between Min_Data = 1 and Max_Data = 4.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
  – The sample time value to be set for the selected channel. This parameter can be a value of ***ADC_sampling_times***
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
  – Defines the offset to be subtracted from the raw converted data when convert injected channels. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
  – Specifies the number of ADC conversions that will be done using the sequencer for injected channel group. This parameter must be a number between Min_Data = 1 and Max_Data = 4.
- ***uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv***
  – Enables or disables the selected ADC automatic injected group conversion after regular one
- ***uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
  – Specifies whether the conversion is performed in Discontinuous mode or not for injected channels. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
  – Select the external trigger edge and enable the trigger of an injected channels. This parameter can be a value of ***ADCEx_External_trigger_edge_Injected***
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***

- Select the external event used to trigger the start of conversion of a injected channels. This parameter can be a value of ***ADCEx_External_trigger_Source_Injected***

5.1.2 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 ***ADCEx_delay_between_2_sampling_phases***

5.2 ADCEx Firmware driver API description

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

5.2.1 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 __ADC_CLK_ENABLE()
   b. ADC pins configuration
      - Enable the clock for the ADC GPIOs using the following function: __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 __DMAx_CLK_ENABLE()
      - Configure and enable two DMA streams stream for managing data transfer from peripheral to memory (output stream)

   - Associate the initilalized 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()

DMA mode IO operation

- Start the ADC peripheral using HAL_ADCEx_InjectedStart_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 ba HAL_ADCEx_InjectedConvCpltCallback() function is executed and user can add his own code by customization of function pointer HAL_ADCEx_InjectedConvCpltCallback
- In case of transfer 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_DMA()

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.

5.2.2 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.
- ***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()***

5.2.3 Extended features functions

5.2.3.1 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**
Notes	• None.

5.2.3.2 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.**
Notes	• None.

5.2.3.3 HAL_ADCEx_InjectedStop

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStop (*ADC_HandleTypeDef ** hadc)**
Function Description	Disables ADC and stop 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.**
Notes	• Caution: This function will stop also regular channels.

5.2.3.4 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**
Notes	• None.

5.2.3.5 HAL_ADCEx_InjectedStop_IT

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (*ADC_HandleTypeDef* * hadc)**
Function Description	Disables the interrupt and stop 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.**
Notes	• Caution: This function will stop also regular channels.

5.2.3.6 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.
Notes	• None.

5.2.3.7 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.

5.2.3.8 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**
Notes	• None.

5.2.3.9 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.**

Notes	• None.

5.2.3.10 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.
Notes	• None.

5.2.3.11 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.
Notes	• None.

5.2.3.12 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**
Notes	• None.

5.3 ADCEx Firmware driver defines

5.3.1 ADCEx

ADCEx

ADCEx_Common_mode

- #define: ***ADC_MODE_INDEPENDENT ((uint32_t)0x00000000)***

- #define: ***ADC_DUALMODE_REGSIMULT_INJECSIMULT ((uint32_t)ADC_CCR_MULTI_0)***

- #define: ***ADC_DUALMODE_REGSIMULT_ALTERTRIG ((uint32_t)ADC_CCR_MULTI_1)***

- #define: ***ADC_DUALMODE_INJECSIMULT ((uint32_t)(ADC_CCR_MULTI_2 | ADC_CCR_MULTI_0))***

- #define: ***ADC_DUALMODE_REGSIMULT ((uint32_t)(ADC_CCR_MULTI_2 | ADC_CCR_MULTI_1))***

- #define: ***ADC_DUALMODE_INTERL ((uint32_t)(ADC_CCR_MULTI_2 | ADC_CCR_MULTI_1 | ADC_CCR_MULTI_0))***

- #define: ***ADC_DUALMODE_ALTERTRIG ((uint32_t)(ADC_CCR_MULTI_3 | ADC_CCR_MULTI_0))***
- #define: ***ADC_TRIPLEMODE_REGSIMULT_INJECSIMULT ((uint32_t)(ADC_CCR_MULTI_4 | ADC_CCR_MULTI_0))***
- #define: ***ADC_TRIPLEMODE_REGSIMULT_AlterTrig ((uint32_t)(ADC_CCR_MULTI_4 | ADC_CCR_MULTI_1))***
- #define: ***ADC_TRIPLEMODE_INJECSIMULT ((uint32_t)(ADC_CCR_MULTI_4 | ADC_CCR_MULTI_2 | ADC_CCR_MULTI_0))***
- #define: ***ADC_TRIPLEMODE_REGSIMULT ((uint32_t)(ADC_CCR_MULTI_4 | ADC_CCR_MULTI_2 | ADC_CCR_MULTI_1))***
- #define: ***ADC_TRIPLEMODE_INTERL ((uint32_t)(ADC_CCR_MULTI_4 | ADC_CCR_MULTI_2 | ADC_CCR_MULTI_1 | ADC_CCR_MULTI_0))***
- #define: ***ADC_TRIPLEMODE_ALTERTRIG ((uint32_t)(ADC_CCR_MULTI_4 | ADC_CCR_MULTI_3 | ADC_CCR_MULTI_0))***

ADCEx_delay_between_2_sampling_phases

- #define: ***ADC_TWOSAMPLINGDELAY_5CYCLES ((uint32_t)0x00000000)***
- #define: ***ADC_TWOSAMPLINGDELAY_6CYCLES ((uint32_t)ADC_CCR_DELAY_0)***
- #define: ***ADC_TWOSAMPLINGDELAY_7CYCLES ((uint32_t)ADC_CCR_DELAY_1)***

- #define: ***ADC_TWOSAMPLINGDELAY_8CYCLES ((uint32_t)(ADC_CCR_DELAY_1 | ADC_CCR_DELAY_0))***
- #define: ***ADC_TWOSAMPLINGDELAY_9CYCLES ((uint32_t)ADC_CCR_DELAY_2)***
- #define: ***ADC_TWOSAMPLINGDELAY_10CYCLES ((uint32_t)(ADC_CCR_DELAY_2 | ADC_CCR_DELAY_0))***
- #define: ***ADC_TWOSAMPLINGDELAY_11CYCLES ((uint32_t)(ADC_CCR_DELAY_2 | ADC_CCR_DELAY_1))***
- #define: ***ADC_TWOSAMPLINGDELAY_12CYCLES ((uint32_t)(ADC_CCR_DELAY_2 | ADC_CCR_DELAY_1 | ADC_CCR_DELAY_0))***
- #define: ***ADC_TWOSAMPLINGDELAY_13CYCLES ((uint32_t)ADC_CCR_DELAY_3)***
- #define: ***ADC_TWOSAMPLINGDELAY_14CYCLES ((uint32_t)(ADC_CCR_DELAY_3 | ADC_CCR_DELAY_0))***
- #define: ***ADC_TWOSAMPLINGDELAY_15CYCLES ((uint32_t)(ADC_CCR_DELAY_3 | ADC_CCR_DELAY_1))***
- #define: ***ADC_TWOSAMPLINGDELAY_16CYCLES ((uint32_t)(ADC_CCR_DELAY_3 | ADC_CCR_DELAY_1 | ADC_CCR_DELAY_0))***
- #define: ***ADC_TWOSAMPLINGDELAY_17CYCLES ((uint32_t)(ADC_CCR_DELAY_3 | ADC_CCR_DELAY_2))***

- #define: ***ADC_TWOSAMPLINGDELAY_18CYCLES ((uint32_t)(ADC_CCR_DELAY_3 | ADC_CCR_DELAY_2 | ADC_CCR_DELAY_0))***

- #define: ***ADC_TWOSAMPLINGDELAY_19CYCLES ((uint32_t)(ADC_CCR_DELAY_3 | ADC_CCR_DELAY_2 | ADC_CCR_DELAY_1))***

- #define: ***ADC_TWOSAMPLINGDELAY_20CYCLES ((uint32_t)ADC_CCR_DELAY)***

ADCEx_Direct_memory_access_mode_for_multi_mode

- #define: ***ADC_DMAACCESSMODE_DISABLED ((uint32_t)0x00000000)***

DMA mode disabled

- #define: ***ADC_DMAACCESSMODE_1 ((uint32_t)ADC_CCR_DMA_0)***

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

- #define: ***ADC_DMAACCESSMODE_2 ((uint32_t)ADC_CCR_DMA_1)***

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

- #define: ***ADC_DMAACCESSMODE_3 ((uint32_t)ADC_CCR_DMA)***

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

ADCEx_External_trigger_edge_Injected

- #define: ***ADC_EXTERNALTRIGINJECCONVEDGE_NONE ((uint32_t)0x00000000)***

- #define: ***ADC_EXTERNALTRIGINJECCONVEDGE_RISING ((uint32_t)ADC_CR2_JEXTEN_0)***

- #define: ***ADC_EXTERNALTRIGINJECCONVEDGE_FALLING ((uint32_t)ADC_CR2_JEXTEN_1)***

- #define: ***ADC_EXTERNALTRIGINJECCONVEDGE_RISINGFALLING ((uint32_t)ADC_CR2_JEXTEN)***

ADCEx_External_trigger_Source_Injected

- #define: ***ADC_EXTERNALTRIGINJECCONV_T1_CC4 ((uint32_t)0x00000000)***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T1_TRGO ((uint32_t)ADC_CR2_JEXTSEL_0)***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T2_CC1 ((uint32_t)ADC_CR2_JEXTSEL_1)***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T2_TRGO ((uint32_t)(ADC_CR2_JEXTSEL_1 | ADC_CR2_JEXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T3_CC2 ((uint32_t)ADC_CR2_JEXTSEL_2)***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T3_CC4 ((uint32_t)(ADC_CR2_JEXTSEL_2 | ADC_CR2_JEXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T4_CC1 ((uint32_t)(ADC_CR2_JEXTSEL_2 | ADC_CR2_JEXTSEL_1))***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T4_CC2 ((uint32_t)(ADC_CR2_JEXTSEL_2 | ADC_CR2_JEXTSEL_1 | ADC_CR2_JEXTSEL_0))***
- #define: ***ADC_EXTERNALTRIGINJECCONV_T4_CC3 ((uint32_t)ADC_CR2_JEXTSEL_3)***

- #define: ***ADC_EXTERNALTRIGINJECCONV_T4_TRGO ((uint32_t)(ADC_CR2_JEXTSEL_3 | ADC_CR2_JEXTSEL_0))***

- #define: ***ADC_EXTERNALTRIGINJECCONV_T5_CC4 ((uint32_t)(ADC_CR2_JEXTSEL_3 | ADC_CR2_JEXTSEL_1))***

- #define: ***ADC_EXTERNALTRIGINJECCONV_T5_TRGO ((uint32_t)(ADC_CR2_JEXTSEL_3 | ADC_CR2_JEXTSEL_1 | ADC_CR2_JEXTSEL_0))***

- #define: ***ADC_EXTERNALTRIGINJECCONV_T8_CC2 ((uint32_t)(ADC_CR2_JEXTSEL_3 | ADC_CR2_JEXTSEL_2))***

- #define: ***ADC_EXTERNALTRIGINJECCONV_T8_CC3 ((uint32_t)(ADC_CR2_JEXTSEL_3 | ADC_CR2_JEXTSEL_2 | ADC_CR2_JEXTSEL_0))***

- #define: ***ADC_EXTERNALTRIGINJECCONV_T8_CC4 ((uint32_t)(ADC_CR2_JEXTSEL_3 | ADC_CR2_JEXTSEL_2 | ADC_CR2_JEXTSEL_1))***

- #define: ***ADC_EXTERNALTRIGINJECCONV_EXT_IT15 ((uint32_t)ADC_CR2_JEXTSEL)***

ADCEx_injected_channel_selection

- #define: ***ADC_INJECTED_RANK_1 ((uint32_t)0x00000001)***

- #define: ***ADC_INJECTED_RANK_2 ((uint32_t)0x00000002)***

- #define: ***ADC_INJECTED_RANK_3 ((uint32_t)0x00000003)***

- #define: ***ADC_INJECTED_RANK_4 ((uint32_t)0x00000004)***

6 HAL CAN Generic Driver

6.1 CAN Firmware driver registers structures

6.1.1 CAN_HandleTypeDef

CAN_HandleTypeDef is defined in the stm32f4xx_hal_can.h

Data Fields

- ***CAN_TypeDef * Instance***
- ***CAN_InitTypeDef Init***
- ***CanTxMsgTypeDef * pTxMsg***
- ***CanRxMsgTypeDef * pRxMsg***
- ***__IO HAL_CAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***CAN_TypeDef* CAN_HandleTypeDef::Instance***
  - Register base address
- ***CAN_InitTypeDef CAN_HandleTypeDef::Init***
  - CAN required parameters
- ***CanTxMsgTypeDef* CAN_HandleTypeDef::pTxMsg***
  - Pointer to transmit structure
- ***CanRxMsgTypeDef* CAN_HandleTypeDef::pRxMsg***
  - Pointer to reception structure
- ***__IO HAL_CAN_StateTypeDef CAN_HandleTypeDef::State***
  - CAN communication state
- ***HAL_LockTypeDef CAN_HandleTypeDef::Lock***
  - CAN locking object
- ***__IO uint32_t CAN_HandleTypeDef::ErrorCode***
  - CAN Error code

6.1.2 CAN_InitTypeDef

CAN_InitTypeDef is defined in the stm32f4xx_hal_can.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Mode***
- ***uint32_t SJW***
- ***uint32_t BS1***
- ***uint32_t BS2***
- ***uint32_t TTCM***

- ***uint32_t ABOM***
- ***uint32_t AWUM***
- ***uint32_t NART***
- ***uint32_t RFLM***
- ***uint32_t TXFP***

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::SJW***
  - 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::BS1***
  - 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::BS2***
  - Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of ***CAN_time_quantum_in_bit_segment_2***
- ***uint32_t CAN_InitTypeDef::TTCM***
  - Enable or disable the time triggered communication mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_InitTypeDef::ABOM***
  - Enable or disable the automatic bus-off management. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::AWUM***
  - Enable or disable the automatic wake-up mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::NART***
  - Enable or disable the non-automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::RFLM***
  - Enable or disable the receive FIFO Locked mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t CAN_InitTypeDef::TXFP***
  - Enable or disable the transmit FIFO priority. This parameter can be set to ENABLE or DISABLE

6.1.3 CAN_FilterConfTypeDef

CAN_FilterConfTypeDef 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 FilterNumber***
- ***uint32_t FilterMode***
- ***uint32_t FilterScale***
- ***uint32_t FilterActivation***
- ***uint32_t BankNumber***

Field Documentation

- ***uint32_t CAN_FilterConfTypeDef::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_FilterConfTypeDef::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_FilterConfTypeDef::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_FilterConfTypeDef::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_FilterConfTypeDef::FilterFIFOAssignment***
  – Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of ***CAN_filter_FIFO***
- ***uint32_t CAN_FilterConfTypeDef::FilterNumber***
  – Specifies the filter which will be initialized. This parameter must be a number between Min_Data = 0 and Max_Data = 27
- ***uint32_t CAN_FilterConfTypeDef::FilterMode***
  – Specifies the filter mode to be initialized. This parameter can be a value of ***CAN_filter_mode***
- ***uint32_t CAN_FilterConfTypeDef::FilterScale***
  – Specifies the filter scale. This parameter can be a value of ***CAN_filter_scale***
- ***uint32_t CAN_FilterConfTypeDef::FilterActivation***
  – Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_FilterConfTypeDef::BankNumber***
  – Select the start slave bank filter. This parameter must be a number between Min_Data = 0 and Max_Data = 28

6.1.4 CAN_FIFOMailBox_TypeDef

CAN_FIFOMailBox_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t RIR***
- ***__IO uint32_t RDTR***
- ***__IO uint32_t RDLR***
- ***__IO uint32_t RDHR***

Field Documentation

- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RIR***
  – CAN receive FIFO mailbox identifier register
- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RDTR***
  – CAN receive FIFO mailbox data length control and time stamp register
- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RDLR***
  – CAN receive FIFO mailbox data low register
- ***__IO uint32_t CAN_FIFOMailBox_TypeDef::RDHR***
  – CAN receive FIFO mailbox data high register

6.1.5 CAN_FilterRegister_TypeDef

CAN_FilterRegister_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t FR1***
- ***__IO uint32_t FR2***

Field Documentation

- ***__IO uint32_t CAN_FilterRegister_TypeDef::FR1***
  – CAN Filter bank register 1
- ***__IO uint32_t CAN_FilterRegister_TypeDef::FR2***
  – CAN Filter bank register 1

6.1.6 CAN_TxMailBox_TypeDef

CAN_TxMailBox_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t TIR***
- ***__IO uint32_t TDTR***
- ***__IO uint32_t TDLR***
- ***__IO uint32_t TDHR***

Field Documentation

- ***__IO uint32_t CAN_TxMailBox_TypeDef::TIR***
  - CAN TX mailbox identifier register
- ***__IO uint32_t CAN_TxMailBox_TypeDef::TDTR***
  - CAN mailbox data length control and time stamp register
- ***__IO uint32_t CAN_TxMailBox_TypeDef::TDLR***
  - CAN mailbox data low register
- ***__IO uint32_t CAN_TxMailBox_TypeDef::TDHR***
  - CAN mailbox data high register

6.1.7 CAN_TypeDef

CAN_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t MCR***
- ***__IO uint32_t MSR***
- ***__IO uint32_t TSR***
- ***__IO uint32_t RF0R***
- ***__IO uint32_t RF1R***
- ***__IO uint32_t IER***
- ***__IO uint32_t ESR***
- ***__IO uint32_t BTR***
- ***uint32_t RESERVED0***
- ***CAN_TxMailBox_TypeDef sTxMailBox***
- ***CAN_FIFOMailBox_TypeDef sFIFOMailBox***
- ***uint32_t RESERVED1***
- ***__IO uint32_t FMR***
- ***__IO uint32_t FM1R***
- ***uint32_t RESERVED2***
- ***__IO uint32_t FS1R***
- ***uint32_t RESERVED3***
- ***__IO uint32_t FFA1R***
- ***uint32_t RESERVED4***
- ***__IO uint32_t FA1R***
- ***uint32_t RESERVED5***
- ***CAN_FilterRegister_TypeDef sFilterRegister***

Field Documentation

- ***__IO uint32_t CAN_TypeDef::MCR***
  - CAN master control register, Address offset: 0x00
- ***__IO uint32_t CAN_TypeDef::MSR***
  - CAN master status register, Address offset: 0x04
- ***__IO uint32_t CAN_TypeDef::TSR***
  - CAN transmit status register, Address offset: 0x08
- ***__IO uint32_t CAN_TypeDef::RF0R***

- CAN receive FIFO 0 register, Address offset: 0x0C
- ***__IO uint32_t CAN_TypeDef::RF1R***
  - CAN receive FIFO 1 register, Address offset: 0x10
- ***__IO uint32_t CAN_TypeDef::IER***
  - CAN interrupt enable register, Address offset: 0x14
- ***__IO uint32_t CAN_TypeDef::ESR***
  - CAN error status register, Address offset: 0x18
- ***__IO uint32_t CAN_TypeDef::BTR***
  - CAN bit timing register, Address offset: 0x1C
- ***uint32_t CAN_TypeDef::RESERVED0[88]***
  - Reserved, 0x020 - 0x17F
- ***CAN_TxMailBox_TypeDef CAN_TypeDef::sTxMailBox[3]***
  - CAN Tx MailBox, Address offset: 0x180 - 0x1AC
- ***CAN_FIFOMailBox_TypeDef CAN_TypeDef::sFIFOMailBox[2]***
  - CAN FIFO MailBox, Address offset: 0x1B0 - 0x1CC
- ***uint32_t CAN_TypeDef::RESERVED1[12]***
  - Reserved, 0x1D0 - 0x1FF
- ***__IO uint32_t CAN_TypeDef::FMR***
  - CAN filter master register, Address offset: 0x200
- ***__IO uint32_t CAN_TypeDef::FM1R***
  - CAN filter mode register, Address offset: 0x204
- ***uint32_t CAN_TypeDef::RESERVED2***
  - Reserved, 0x208
- ***__IO uint32_t CAN_TypeDef::FS1R***
  - CAN filter scale register, Address offset: 0x20C
- ***uint32_t CAN_TypeDef::RESERVED3***
  - Reserved, 0x210
- ***__IO uint32_t CAN_TypeDef::FFA1R***
  - CAN filter FIFO assignment register, Address offset: 0x214
- ***uint32_t CAN_TypeDef::RESERVED4***
  - Reserved, 0x218
- ***__IO uint32_t CAN_TypeDef::FA1R***
  - CAN filter activation register, Address offset: 0x21C
- ***uint32_t CAN_TypeDef::RESERVED5[8]***
  - Reserved, 0x220-0x23F
- ***CAN_FilterRegister_TypeDef CAN_TypeDef::sFilterRegister[28]***
  - CAN Filter Register, Address offset: 0x240-0x31C

6.2 CAN Firmware driver API description

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

6.2.1 How to use this driver

1. Enable the CAN controller interface clock using __CAN1_CLK_ENABLE() for CAN1 and __CAN1_CLK_ENABLE() for CAN2 In case you are using CAN2 only, you have to enable the CAN1 clock.
2. CAN pins configuration

  - Enable the clock for the CAN GPIOs using the following function: __GPIOx_CLK_ENABLE()
  - Connect and configure the involved CAN pins to AF9 using the following function HAL_GPIO_Init()
3. Initialise and configure the CAN using CAN_Init() function.
4. Transmit the desired CAN frame using HAL_CAN_Transmit() function.
5. Receive a CAN frame using HAL_CAN_Recieve() function.

Polling mode IO operation

- Start the CAN peripheral transmission and wait the end of this operation using HAL_CAN_Transmit(), at this stage user can specify the value of timeout according to his end application
- Start the CAN peripheral reception and wait the end of this operation using HAL_CAN_Receive(), at this stage user can specify the value of timeout according to his end application

Interrupt mode IO operation

- Start the CAN peripheral transmission using HAL_CAN_Transmit_IT()
- Start the CAN peripheral reception using HAL_CAN_Receive_IT()
- Use HAL_CAN_IRQHandler() called under the used CAN Interrupt subroutine
- At CAN end of transmission HAL_CAN_TxCpltCallback() function is executed and user can add his own code by customization of function pointer HAL_CAN_TxCpltCallback
- In case of CAN Error, HAL_CAN_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_CAN_ErrorCallback

CAN HAL driver macros list

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

- __HAL_CAN_ENABLE_IT: Enable the specified CAN interrupts
- __HAL_CAN_DISABLE_IT: Disable the specified CAN interrupts
- __HAL_CAN_GET_IT_SOURCE: Check if the specified CAN interrupt source is enabled or disabled
- __HAL_CAN_CLEAR_FLAG: Clear the CAN's pending flags
- __HAL_CAN_GET_FLAG: Get the selected CAN's flag status

You can refer to the CAN HAL driver header file for more useful macros

6.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the CAN.
- De-initialize the CAN.
- ***HAL_CAN_Init()***
- ***HAL_CAN_ConfigFilter()***

- ***HAL_CAN_DeInit()***
- ***HAL_CAN_MspInit()***
- ***HAL_CAN_MspDeInit()***

6.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a CAN frame message.
- Receive a CAN frame message.
- Enter CAN peripheral in sleep mode.
- Wake up the CAN peripheral from sleep mode.
- ***HAL_CAN_Transmit()***
- ***HAL_CAN_Transmit_IT()***
- ***HAL_CAN_Receive()***
- ***HAL_CAN_Receive_IT()***
- ***HAL_CAN_Sleep()***
- ***HAL_CAN_WakeUp()***
- ***HAL_CAN_IRQHandler()***
- ***HAL_CAN_TxCpltCallback()***
- ***HAL_CAN_RxCpltCallback()***
- ***HAL_CAN_ErrorCallback()***

6.2.4 Peripheral State and Error functions

This subsection provides functions allowing to :

- Check the CAN state.
- Check CAN Errors detected during interrupt process
- ***HAL_CAN_GetState()***
- ***HAL_CAN_GetError()***

6.2.5 Initialization and de-initialization functions

6.2.5.1 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**
Notes	• None.

6.2.5.2 HAL_CAN_ConfigFilter

Function Name	**HAL_StatusTypeDef HAL_CAN_ConfigFilter (*CAN_HandleTypeDef ** hcan, *CAN_FilterConfTypeDef ** 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_FilterConfTypeDef structure that contains the filter configuration information.
Return values	• None.
Notes	• None.

6.2.5.3 HAL_CAN_DeInit

Function Name	**HAL_StatusTypeDef HAL_CAN_DeInit (*CAN_HandleTypeDef ** hcan)**
Function Description	Deinitializes the CANx 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**
Notes	• None.

6.2.5.4 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.

Notes	• None.

6.2.5.5 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.
Notes	• None.

6.2.6 IO operation functions

6.2.6.1 HAL_CAN_Transmit

Function Name	**HAL_StatusTypeDef HAL_CAN_Transmit (*CAN_HandleTypeDef* * hcan, uint32_t Timeout)**
Function Description	Initiates and transmits a CAN frame message.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

6.2.6.2 HAL_CAN_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_CAN_Transmit_IT (*CAN_HandleTypeDef* * hcan)**

Function Description	Initiates and transmits a CAN frame message.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL status**
Notes	• None.

6.2.6.3 HAL_CAN_Receive

Function Name	**HAL_StatusTypeDef HAL_CAN_Receive (*CAN_HandleTypeDef ** hcan, uint8_t FIFONumber, uint32_t Timeout)**
Function Description	Receives a correct CAN frame.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber :** FIFO Number value • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

6.2.6.4 HAL_CAN_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_CAN_Receive_IT (*CAN_HandleTypeDef ** hcan, uint8_t FIFONumber)**
Function Description	Receives a correct CAN frame.
Parameters	• **hcan :** Pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber :** Specify the FIFO number
Return values	• **HAL status**
Notes	• None.

6.2.6.5 HAL_CAN_Sleep

Function Name	**HAL_StatusTypeDef HAL_CAN_Sleep (*CAN_HandleTypeDef ** hcan)**
Function Description	Enters the Sleep (low power) mode.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL status.**
Notes	• None.

6.2.6.6 HAL_CAN_WakeUp

Function Name	**HAL_StatusTypeDef HAL_CAN_WakeUp (*CAN_HandleTypeDef ** hcan)**
Function Description	Wakes up the CAN peripheral from sleep mode, after that the CAN peripheral is in the normal mode.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• **HAL status.**
Notes	• None.

6.2.6.7 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.
Notes	• None.

6.2.6.8 HAL_CAN_TxCpltCallback

Function Name	**void HAL_CAN_TxCpltCallback (** ***CAN_HandleTypeDef **** **hcan)**
Function Description	Transmission complete callback in non blocking mode.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• None.
Notes	• None.

6.2.6.9 HAL_CAN_RxCpltCallback

Function Name	**void HAL_CAN_RxCpltCallback (** ***CAN_HandleTypeDef **** **hcan)**
Function Description	Transmission complete callback in non blocking mode.
Parameters	• **hcan :** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• None.
Notes	• None.

6.2.6.10 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.
Notes	• None.

6.2.7 Peripheral State and Error functions

6.2.7.1 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**
Notes	• None.

6.2.7.2 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**
Notes	• None.

6.3 CAN Firmware driver defines

6.3.1 CAN

CAN

CAN_Exported_Constants

- #define: ***INAK_TIMEOUT ((uint32_t)0x0000FFFF)***

- #define: ***SLAK_TIMEOUT ((uint32_t)0x0000FFFF)***

- #define: ***CAN_TXMAILBOX_0 ((uint8_t)0x00)***

- #define: ***CAN_TXMAILBOX_1 ((uint8_t)0x01)***

- #define: ***CAN_TXMAILBOX_2 ((uint8_t)0x02)***

CAN_filter_FIFO

- #define: ***CAN_FILTER_FIFO0 ((uint8_t)0x00)***

Filter FIFO 0 assignment for filter x

- #define: ***CAN_FILTER_FIFO1 ((uint8_t)0x01)***

Filter FIFO 1 assignment for filter x

- #define: ***CAN_FilterFIFO0 CAN_FILTER_FIFO0***

- #define: ***CAN_FilterFIFO1 CAN_FILTER_FIFO1***

CAN_filter_mode

- #define: ***CAN_FILTERMODE_IDMASK ((uint8_t)0x00)***

Identifier mask mode

- #define: ***CAN_FILTERMODE_IDLIST ((uint8_t)0x01)***

Identifier list mode

CAN_filter_scale

- #define: ***CAN_FILTERSCALE_16BIT ((uint8_t)0x00)***

Two 16-bit filters

- #define: ***CAN_FILTERSCALE_32BIT ((uint8_t)0x01)***

One 32-bit filter

CAN_flags

- #define: ***CAN_FLAG_RQCP0 ((uint32_t)0x00000500)***

Request MailBox0 flag

- #define: ***CAN_FLAG_RQCP1 ((uint32_t)0x00000508)***

Request MailBox1 flag

- #define: ***CAN_FLAG_RQCP2 ((uint32_t)0x00000510)***

Request MailBox2 flag

- #define: ***CAN_FLAG_TXOK0 ((uint32_t)0x00000501)***

Transmission OK MailBox0 flag

- #define: ***CAN_FLAG_TXOK1 ((uint32_t)0x00000509)***

Transmission OK MailBox1 flag

- #define: ***CAN_FLAG_TXOK2 ((uint32_t)0x00000511)***

Transmission OK MailBox2 flag

- #define: ***CAN_FLAG_TME0 ((uint32_t)0x0000051A)***

Transmit mailbox 0 empty flag

- #define: ***CAN_FLAG_TME1 ((uint32_t)0x0000051B)***

Transmit mailbox 0 empty flag

- #define: ***CAN_FLAG_TME2 ((uint32_t)0x0000051C)***

Transmit mailbox 0 empty flag

- #define: ***CAN_FLAG_FF0 ((uint32_t)0x00000203)***

FIFO 0 Full flag

- #define: ***CAN_FLAG_FOV0 ((uint32_t)0x00000204)***

FIFO 0 Overrun flag

- #define: ***CAN_FLAG_FF1 ((uint32_t)0x00000403)***

FIFO 1 Full flag

- #define: ***CAN_FLAG_FOV1 ((uint32_t)0x00000404)***

FIFO 1 Overrun flag

- #define: ***CAN_FLAG_WKU ((uint32_t)0x00000103)***

Wake up flag

- #define: ***CAN_FLAG_SLAK ((uint32_t)0x00000101)***

Sleep acknowledge flag

- #define: ***CAN_FLAG_SLAKI ((uint32_t)0x00000104)***

Sleep acknowledge flag

- #define: ***CAN_FLAG_EWG ((uint32_t)0x00000300)***

Error warning flag

- #define: ***CAN_FLAG_EPV ((uint32_t)0x00000301)***

Error passive flag

- #define: ***CAN_FLAG_BOF ((uint32_t)0x00000302)***

Bus-Off flag

CAN_identifier_type

- #define: ***CAN_ID_STD ((uint32_t)0x00000000)***

Standard Id

- #define: ***CAN_ID_EXT ((uint32_t)0x00000004)***

Extended Id

CAN_InitStatus

- #define: ***CAN_INITSTATUS_FAILED ((uint8_t)0x00)***

CAN initialization failed

- #define: ***CAN_INITSTATUS_SUCCESS ((uint8_t)0x01)***

CAN initialization OK

CAN_interrupts

- #define: ***CAN_IT_TME ((uint32_t)CAN_IER_TMEIE)***

Transmit mailbox empty interrupt

- #define: ***CAN_IT_FMP0 ((uint32_t)CAN_IER_FMPIE0)***

FIFO 0 message pending interrupt

- #define: ***CAN_IT_FF0 ((uint32_t)CAN_IER_FFIE0)***

FIFO 0 full interrupt

- #define: ***CAN_IT_FOV0 ((uint32_t)CAN_IER_FOVIE0)***

FIFO 0 overrun interrupt

- #define: ***CAN_IT_FMP1 ((uint32_t)CAN_IER_FMPIE1)***

FIFO 1 message pending interrupt

- #define: ***CAN_IT_FF1 ((uint32_t)CAN_IER_FFIE1)***

FIFO 1 full interrupt

- #define: ***CAN_IT_FOV1 ((uint32_t)CAN_IER_FOVIE1)***

FIFO 1 overrun interrupt

- #define: ***CAN_IT_WKU ((uint32_t)CAN_IER_WKUIE)***

Wake-up interrupt

- #define: ***CAN_IT_SLK ((uint32_t)CAN_IER_SLKIE)***

Sleep acknowledge interrupt

- #define: ***CAN_IT_EWG ((uint32_t)CAN_IER_EWGIE)***

Error warning interrupt

- #define: ***CAN_IT_EPV ((uint32_t)CAN_IER_EPVIE)***

Error passive interrupt

- #define: ***CAN_IT_BOF ((uint32_t)CAN_IER_BOFIE)***

Bus-off interrupt

- #define: ***CAN_IT_LEC ((uint32_t)CAN_IER_LECIE)***

Last error code interrupt

- #define: ***CAN_IT_ERR ((uint32_t)CAN_IER_ERRIE)***

Error Interrupt

- #define: ***CAN_IT_RQCP0 CAN_IT_TME***

- #define: ***CAN_IT_RQCP1 CAN_IT_TME***

- #define: ***CAN_IT_RQCP2 CAN_IT_TME***

CAN_operating_mode

- #define: ***CAN_MODE_NORMAL ((uint32_t)0x00000000)***

Normal mode

- #define: ***CAN_MODE_LOOPBACK ((uint32_t)CAN_BTR_LBKM)***

Loopback mode

- #define: ***CAN_MODE_SILENT ((uint32_t)CAN_BTR_SILM)***

Silent mode

- #define: ***CAN_MODE_SILENT_LOOPBACK ((uint32_t)(CAN_BTR_LBKM | CAN_BTR_SILM))***

Loopback combined with silent mode

CAN_receive_FIFO_number_constants

- #define: ***CAN_FIFO0 ((uint8_t)0x00)***

CAN FIFO 0 used to receive

- #define: ***CAN_FIFO1 ((uint8_t)0x01)***

CAN FIFO 1 used to receive

CAN_remote_transmission_request

- #define: ***CAN_RTR_DATA ((uint32_t)0x00000000)***

Data frame

- #define: ***CAN_RTR_REMOTE ((uint32_t)0x00000002)***

Remote frame

CAN_synchronisation_jump_width

- #define: ***CAN_SJW_1TQ ((uint32_t)0x00000000)***

1 time quantum

- #define: ***CAN_SJW_2TQ ((uint32_t)CAN_BTR_SJW_0)***

2 time quantum

- #define: ***CAN_SJW_3TQ ((uint32_t)CAN_BTR_SJW_1)***

3 time quantum

- #define: ***CAN_SJW_4TQ ((uint32_t)CAN_BTR_SJW)***

4 time quantum

CAN_time_quantum_in_bit_segment_1

- #define: ***CAN_BS1_1TQ ((uint32_t)0x00000000)***

1 time quantum

- #define: ***CAN_BS1_2TQ ((uint32_t)CAN_BTR_TS1_0)***

2 time quantum

- #define: ***CAN_BS1_3TQ ((uint32_t)CAN_BTR_TS1_1)***

3 time quantum

- #define: ***CAN_BS1_4TQ ((uint32_t)(CAN_BTR_TS1_1 | CAN_BTR_TS1_0))***

4 time quantum

- #define: ***CAN_BS1_5TQ ((uint32_t)CAN_BTR_TS1_2)***

5 time quantum

- #define: ***CAN_BS1_6TQ ((uint32_t)(CAN_BTR_TS1_2 | CAN_BTR_TS1_0))***

6 time quantum

- #define: ***CAN_BS1_7TQ ((uint32_t)(CAN_BTR_TS1_2 | CAN_BTR_TS1_1))***

7 time quantum

- #define: ***CAN_BS1_8TQ ((uint32_t)(CAN_BTR_TS1_2 | CAN_BTR_TS1_1 | CAN_BTR_TS1_0))***

8 time quantum

- #define: ***CAN_BS1_9TQ ((uint32_t)CAN_BTR_TS1_3)***

9 time quantum

- #define: ***CAN_BS1_10TQ ((uint32_t)(CAN_BTR_TS1_3 | CAN_BTR_TS1_0))***

10 time quantum

- #define: ***CAN_BS1_11TQ ((uint32_t)(CAN_BTR_TS1_3 | CAN_BTR_TS1_1))***

11 time quantum

- #define: ***CAN_BS1_12TQ ((uint32_t)(CAN_BTR_TS1_3 | CAN_BTR_TS1_1 | CAN_BTR_TS1_0))***

12 time quantum

- #define: ***CAN_BS1_13TQ ((uint32_t)(CAN_BTR_TS1_3 | CAN_BTR_TS1_2))***

13 time quantum

- #define: ***CAN_BS1_14TQ ((uint32_t)(CAN_BTR_TS1_3 | CAN_BTR_TS1_2 | CAN_BTR_TS1_0))***

14 time quantum

- #define: ***CAN_BS1_15TQ ((uint32_t)(CAN_BTR_TS1_3 | CAN_BTR_TS1_2 | CAN_BTR_TS1_1))***

15 time quantum

- #define: ***CAN_BS1_16TQ ((uint32_t)CAN_BTR_TS1)***

16 time quantum

CAN_time_quantum_in_bit_segment_2

- #define: ***CAN_BS2_1TQ ((uint32_t)0x00000000)***

1 time quantum

- #define: ***CAN_BS2_2TQ ((uint32_t)CAN_BTR_TS2_0)***

2 time quantum

- #define: ***CAN_BS2_3TQ ((uint32_t)CAN_BTR_TS2_1)***

3 time quantum

- #define: ***CAN_BS2_4TQ ((uint32_t)(CAN_BTR_TS2_1 | CAN_BTR_TS2_0))***

4 time quantum

- #define: ***CAN_BS2_5TQ ((uint32_t)CAN_BTR_TS2_2)***

5 time quantum

- #define: ***CAN_BS2_6TQ ((uint32_t)(CAN_BTR_TS2_2 | CAN_BTR_TS2_0))***

6 time quantum

- #define: ***CAN_BS2_7TQ ((uint32_t)(CAN_BTR_TS2_2 | CAN_BTR_TS2_1))***

7 time quantum

- #define: ***CAN_BS2_8TQ ((uint32_t)CAN_BTR_TS2)***

8 time quantum

CAN_transmit_constants

- #define: ***CAN_TXSTATUS_FAILED ((uint8_t)0x00)***

CAN transmission failed

- #define: ***CAN_TXSTATUS_OK ((uint8_t)0x01)***

CAN transmission succeeded

- #define: ***CAN_TXSTATUS_PENDING ((uint8_t)0x02)***

CAN transmission pending

- #define: ***CAN_TXSTATUS_NOMAILBOX ((uint8_t)0x04)***

CAN cell did not provide CAN_TxStatus_NoMailBox

7 HAL CORTEX Generic Driver

7.1 CORTEX Firmware driver API description

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

7.1.1 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. Refer to STM32F3xx/STM32F4xx Cortex-M4 programming manual (PM0214) for details.
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 programing manual for details in how to configure priority. When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority. IRQ priority order (sorted by highest to lowest priority): Lowest pre-emption priority Lowest sub priority Lowest hardware priority (IRQ number)

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for 1 msec interrupts.

- 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

7.1.2 Initialization and de-initialization functions

This section provides the CORTEX HAL driver functions allowing to configure Interrupts Systick functionalities

- *HAL_NVIC_SetPriorityGrouping()*
- *HAL_NVIC_SetPriority()*
- *HAL_NVIC_EnableIRQ()*
- *HAL_NVIC_DisableIRQ()*
- *HAL_NVIC_SystemReset()*
- *HAL_SYSTICK_Config()*

7.1.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK) functionalities.

- *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()*

7.1.4 Initialization and de-initialization functions

7.1.4.1 HAL_NVIC_SetPriorityGrouping

Function Name	**void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)**
Function Description	Sets the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.
Parameters	• **PriorityGroup :** The priority grouping bits length. This parameter can be one of the following values: – *NVIC_PRIORITYGROUP_0 :* 0 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
Return values	• None.
Notes	• When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption

is no more possible. The pending IRQ priority will be managed only by the subpriority.

7.1.4.2 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 stm32f4xx.h file) • **PreemptPriority :** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority • **SubPriority :** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.
Return values	• None.
Notes	• None.

7.1.4.3 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 stm32f4xx.h file)
Return values	• None.
Notes	• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

7.1.4.4 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 stm32f4xx.h file)
Return values	• None.
Notes	• None.

7.1.4.5 HAL_NVIC_SystemReset

Function Name	**void HAL_NVIC_SystemReset (void)**
Function Description	Initiates a system reset request to reset the MCU.
Parameters	• None.
Return values	• None.
Notes	• None.

7.1.4.6 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.***
Notes	• None.

7.1.5 Peripheral Control functions

7.1.5.1 HAL_NVIC_GetPriorityGrouping

Function Name	**uint32_t HAL_NVIC_GetPriorityGrouping (void)**
Function Description	Gets the priority grouping field from the NVIC Interrupt Controller.
Parameters	• None.
Return values	• **Priority grouping field (SCB->AIRCR [10:8] PRIGROUP field)**
Notes	• None.

7.1.5.2 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 stm32f4xx.h file) • **PriorityGroup :** the priority grouping bits length. This parameter can be one of the following values: – ***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 • **pPreemptPriority :** Pointer on the Preemptive priority value (starting from 0). • **pSubPriority :** Pointer on the Subpriority value (starting from 0).

Return values	• None.
Notes	• None.

7.1.5.3 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 stm32f4xx.h file)
Return values	• None.
Notes	• None.

7.1.5.4 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 stm32f4xx.h file)
Return values	• **status : - 0 Interrupt status is not pending.** – ***1 Interrupt status is pending.***
Notes	• None.

7.1.5.5 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 stm32f4xx.h file)
Return values	• None.
Notes	• None.

7.1.5.6 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 stm32f4xx.h file)
Return values	• **status : - 0 Interrupt status is not pending.** – ***1 Interrupt status is pending.***
Notes	• None.

7.1.5.7 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.
Notes	• None.

7.1.5.8 HAL_SYSTICK_IRQHandler

Function Name	**void HAL_SYSTICK_IRQHandler (void)**
Function Description	This function handles SYSTICK interrupt request.
Parameters	• None.
Return values	• None.
Notes	• None.

7.1.5.9 HAL_SYSTICK_Callback

Function Name	**void HAL_SYSTICK_Callback (void)**
Function Description	SYSTICK callback.
Parameters	• None.
Return values	• None.
Notes	• None.

7.2 CORTEX Firmware driver defines

7.2.1 CORTEX

CORTEX

CORTEX_Preemption_Priority_Group

- #define: ***NVIC_PRIORITYGROUP_0 ((uint32_t)0x00000007)***

0 bits for pre-emption priority 4 bits for subpriority

- #define: ***NVIC_PRIORITYGROUP_1 ((uint32_t)0x00000006)***

1 bits for pre-emption priority 3 bits for subpriority

- #define: ***NVIC_PRIORITYGROUP_2 ((uint32_t)0x00000005)***

2 bits for pre-emption priority 2 bits for subpriority

- #define: ***NVIC_PRIORITYGROUP_3 ((uint32_t)0x00000004)***

3 bits for pre-emption priority 1 bits for subpriority

- #define: ***NVIC_PRIORITYGROUP_4 ((uint32_t)0x00000003)***

4 bits for pre-emption priority 0 bits for subpriority

CORTEX_SysTick_clock_source

- #define: ***SYSTICK_CLKSOURCE_HCLK_DIV8 ((uint32_t)0x00000000)***

- #define: ***SYSTICK_CLKSOURCE_HCLK ((uint32_t)0x00000004)***

8 HAL CRC Generic Driver

8.1 CRC Firmware driver registers structures

8.1.1 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

8.1.2 CRC_TypeDef

CRC_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t DR***
- ***__IO uint8_t IDR***
- ***uint8_t RESERVED0***
- ***uint16_t RESERVED1***
- ***__IO uint32_t CR***

Field Documentation

- ***__IO uint32_t CRC_TypeDef::DR***
  - CRC Data register, Address offset: 0x00
- ***__IO uint8_t CRC_TypeDef::IDR***
  - CRC Independent data register, Address offset: 0x04
- ***uint8_t CRC_TypeDef::RESERVED0***
  - Reserved, 0x05
- ***uint16_t CRC_TypeDef::RESERVED1***
  - Reserved, 0x06
- ***__IO uint32_t CRC_TypeDef::CR***

– CRC Control register, Address offset: 0x08

8.2 CRC Firmware driver API description

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

8.2.1 How to use this driver

The CRC HAL driver can be used as follows:

1. Enable CRC AHB clock using __CRC_CLK_ENABLE();
2. Use HAL_CRC_Accumulate() function to compute the CRC value of a 32-bit data buffer using combination of the previous CRC value and the new one.
3. Use HAL_CRC_Calculate() function to compute the CRC Value of a new 32-bit data buffer. This function resets the CRC computation unit before starting the computation to avoid getting wrong CRC values.

8.2.2 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
- DeInitialize CRC MSP
- ***HAL_CRC_Init()***
- ***HAL_CRC_DeInit()***
- ***HAL_CRC_MspInit()***
- ***HAL_CRC_MspDeInit()***

8.2.3 Peripheral Control functions

This section provides functions allowing to:

- Compute the 32-bit CRC value of 32-bit data buffer, using combination of the previous CRC value and the new one.
- Compute the 32-bit CRC value of 32-bit data buffer, independently of the previous CRC value.
- ***HAL_CRC_Accumulate()***
- ***HAL_CRC_Calculate()***

8.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- ***HAL_CRC_GetState()***

8.2.5 Initialization and de-initialization functions

8.2.5.1 HAL_CRC_Init

Function Name	**HAL_StatusTypeDef HAL_CRC_Init (*CRC_HandleTypeDef ** hcrc)**
Function Description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates the associated handle.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• **HAL status**
Notes	• None.

8.2.5.2 HAL_CRC_DeInit

Function Name	**HAL_StatusTypeDef HAL_CRC_DeInit (*CRC_HandleTypeDef ** hcrc)**
Function Description	DeInitializes the CRC peripheral.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• **HAL status**
Notes	• None.

8.2.5.3 HAL_CRC_MspInit

Function Name	**void HAL_CRC_MspInit (*CRC_HandleTypeDef ** hcrc)**
Function Description	Initializes the CRC MSP.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• None.
Notes	• None.

8.2.5.4 HAL_CRC_MspDeInit

Function Name	**void HAL_CRC_MspDeInit (*CRC_HandleTypeDef ** hcrc)**
Function Description	DeInitializes the CRC MSP.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• None.
Notes	• None.

8.2.6 Peripheral Control functions

8.2.6.1 HAL_CRC_Accumulate

Function Name	**uint32_t HAL_CRC_Accumulate (*CRC_HandleTypeDef ** hcrc, uint32_t pBuffer, uint32_t BufferLength)**
Function Description	Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC • **pBuffer :** pointer to the buffer containing the data to be computed • **BufferLength :** length of the buffer to be computed
Return values	• **32-bit CRC**
Notes	• None.

8.2.6.2 HAL_CRC_Calculate

Function Name	**uint32_t HAL_CRC_Calculate (*CRC_HandleTypeDef ** hcrc, uint32_t pBuffer, uint32_t BufferLength)**

Function Description	Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC • **pBuffer :** Pointer to the buffer containing the data to be computed • **BufferLength :** Length of the buffer to be computed
Return values	• **32-bit CRC**
Notes	• None.

8.2.7 Peripheral State functions

8.2.7.1 HAL_CRC_GetState

Function Name	**HAL_CRC_StateTypeDef HAL_CRC_GetState (** ***CRC_HandleTypeDef **** **hcrc)**
Function Description	Returns the CRC state.
Parameters	• **hcrc :** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• **HAL state**
Notes	• None.

8.3 CRC Firmware driver defines

8.3.1 CRC

CRC

9 HAL CRYP Generic Driver

9.1 CRYP Firmware driver registers structures

9.1.1 CRYP_HandleTypeDef

CRYP_HandleTypeDef is defined in the stm32f4xx_hal_cryp.h

Data Fields

- ***CRYP_InitTypeDef Init***
- ***uint8_t * pCrypInBuffPtr***
- ***uint8_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***HAL_StatusTypeDef Status***
- ***HAL_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***

Field Documentation

- ***CRYP_InitTypeDef CRYP_HandleTypeDef::Init***
  - CRYP required parameters
- ***uint8_t* CRYP_HandleTypeDef::pCrypInBuffPtr***
  - Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint8_t* CRYP_HandleTypeDef::pCrypOutBuffPtr***
  - Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t CRYP_HandleTypeDef::CrypInCount***
  - Counter of inputed data
- ***__IO uint16_t CRYP_HandleTypeDef::CrypOutCount***
  - Counter of outputed data
- ***HAL_StatusTypeDef CRYP_HandleTypeDef::Status***
  - CRYP peripheral status
- ***HAL_PhaseTypeDef 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

9.1.2 CRYP_InitTypeDef

CRYP_InitTypeDef is defined in the stm32f4xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint8_t * pKey***
- ***uint8_t * pInitVect***
- ***uint8_t IVSize***
- ***uint8_t TagSize***
- ***uint8_t * Header***
- ***uint16_t HeaderSize***
- ***uint8_t * pScratch***

Field Documentation

- ***uint32_t CRYP_InitTypeDef::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_InitTypeDef::KeySize***
  - Used only in AES mode only : 128, 192 or 256 bit key length. This parameter can be a value of ***CRYP_Key_Size***
- ***uint8_t* CRYP_InitTypeDef::pKey***
  - The key used for encryption/decryption
- ***uint8_t* CRYP_InitTypeDef::pInitVect***
  - The initialization vector used also as initialization counter in CTR mode
- ***uint8_t CRYP_InitTypeDef::IVSize***
  - The size of initialization vector. This parameter (called nonce size in CCM) is used only in AES-128/192/256 encryption/decryption CCM mode
- ***uint8_t CRYP_InitTypeDef::TagSize***
  - The size of returned authentication TAG. This parameter is used only in AES-128/192/256 encryption/decryption CCM mode
- ***uint8_t* CRYP_InitTypeDef::Header***
  - The header used in GCM and CCM modes
- ***uint16_t CRYP_InitTypeDef::HeaderSize***
  - The size of header buffer in bytes
- ***uint8_t* CRYP_InitTypeDef::pScratch***
  - Scratch buffer used to append the header. It's size must be equal to header size + 21 bytes. This parameter is used only in AES-128/192/256 encryption/decryption CCM mode

9.1.3 CRYP_TypeDef

CRYP_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***

- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t DOUT***
- ***__IO uint32_t DMACR***
- ***__IO uint32_t IMSCR***
- ***__IO uint32_t RISR***
- ***__IO uint32_t MISR***
- ***__IO uint32_t K0LR***
- ***__IO uint32_t K0RR***
- ***__IO uint32_t K1LR***
- ***__IO uint32_t K1RR***
- ***__IO uint32_t K2LR***
- ***__IO uint32_t K2RR***
- ***__IO uint32_t K3LR***
- ***__IO uint32_t K3RR***
- ***__IO uint32_t IV0LR***
- ***__IO uint32_t IV0RR***
- ***__IO uint32_t IV1LR***
- ***__IO uint32_t IV1RR***
- ***__IO uint32_t CSGCMCCM0R***
- ***__IO uint32_t CSGCMCCM1R***
- ***__IO uint32_t CSGCMCCM2R***
- ***__IO uint32_t CSGCMCCM3R***
- ***__IO uint32_t CSGCMCCM4R***
- ***__IO uint32_t CSGCMCCM5R***
- ***__IO uint32_t CSGCMCCM6R***
- ***__IO uint32_t CSGCMCCM7R***
- ***__IO uint32_t CSGCM0R***
- ***__IO uint32_t CSGCM1R***
- ***__IO uint32_t CSGCM2R***
- ***__IO uint32_t CSGCM3R***
- ***__IO uint32_t CSGCM4R***
- ***__IO uint32_t CSGCM5R***
- ***__IO uint32_t CSGCM6R***
- ***__IO uint32_t CSGCM7R***

Field Documentation

- ***__IO uint32_t CRYP_TypeDef::CR***
  - CRYP control register, Address offset: 0x00
- ***__IO uint32_t CRYP_TypeDef::SR***
  - CRYP status register, Address offset: 0x04
- ***__IO uint32_t CRYP_TypeDef::DR***
  - CRYP data input register, Address offset: 0x08
- ***__IO uint32_t CRYP_TypeDef::DOUT***
  - CRYP data output register, Address offset: 0x0C
- ***__IO uint32_t CRYP_TypeDef::DMACR***
  - CRYP DMA control register, Address offset: 0x10
- ***__IO uint32_t CRYP_TypeDef::IMSCR***
  - CRYP interrupt mask set/clear register, Address offset: 0x14
- ***__IO uint32_t CRYP_TypeDef::RISR***

  - CRYP raw interrupt status register, Address offset: 0x18
- ***__IO uint32_t CRYP_TypeDef::MISR***
  - CRYP masked interrupt status register, Address offset: 0x1C
- ***__IO uint32_t CRYP_TypeDef::K0LR***
  - CRYP key left register 0, Address offset: 0x20
- ***__IO uint32_t CRYP_TypeDef::K0RR***
  - CRYP key right register 0, Address offset: 0x24
- ***__IO uint32_t CRYP_TypeDef::K1LR***
  - CRYP key left register 1, Address offset: 0x28
- ***__IO uint32_t CRYP_TypeDef::K1RR***
  - CRYP key right register 1, Address offset: 0x2C
- ***__IO uint32_t CRYP_TypeDef::K2LR***
  - CRYP key left register 2, Address offset: 0x30
- ***__IO uint32_t CRYP_TypeDef::K2RR***
  - CRYP key right register 2, Address offset: 0x34
- ***__IO uint32_t CRYP_TypeDef::K3LR***
  - CRYP key left register 3, Address offset: 0x38
- ***__IO uint32_t CRYP_TypeDef::K3RR***
  - CRYP key right register 3, Address offset: 0x3C
- ***__IO uint32_t CRYP_TypeDef::IV0LR***
  - CRYP initialization vector left-word register 0, Address offset: 0x40
- ***__IO uint32_t CRYP_TypeDef::IV0RR***
  - CRYP initialization vector right-word register 0, Address offset: 0x44
- ***__IO uint32_t CRYP_TypeDef::IV1LR***
  - CRYP initialization vector left-word register 1, Address offset: 0x48
- ***__IO uint32_t CRYP_TypeDef::IV1RR***
  - CRYP initialization vector right-word register 1, Address offset: 0x4C
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM0R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 0, Address offset: 0x50
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM1R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 1, Address offset: 0x54
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM2R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 2, Address offset: 0x58
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM3R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 3, Address offset: 0x5C
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM4R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 4, Address offset: 0x60
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM5R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 5, Address offset: 0x64
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM6R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 6, Address offset: 0x68
- ***__IO uint32_t CRYP_TypeDef::CSGCMCCM7R***
  - CRYP GCM/GMAC or CCM/CMAC context swap register 7, Address offset: 0x6C
- ***__IO uint32_t CRYP_TypeDef::CSGCM0R***
  - CRYP GCM/GMAC context swap register 0, Address offset: 0x70

- ***__IO uint32_t CRYP_TypeDef::CSGCM1R***
  - CRYP GCM/GMAC context swap register 1, Address offset: 0x74
- ***__IO uint32_t CRYP_TypeDef::CSGCM2R***
  - CRYP GCM/GMAC context swap register 2, Address offset: 0x78
- ***__IO uint32_t CRYP_TypeDef::CSGCM3R***
  - CRYP GCM/GMAC context swap register 3, Address offset: 0x7C
- ***__IO uint32_t CRYP_TypeDef::CSGCM4R***
  - CRYP GCM/GMAC context swap register 4, Address offset: 0x80
- ***__IO uint32_t CRYP_TypeDef::CSGCM5R***
  - CRYP GCM/GMAC context swap register 5, Address offset: 0x84
- ***__IO uint32_t CRYP_TypeDef::CSGCM6R***
  - CRYP GCM/GMAC context swap register 6, Address offset: 0x88
- ***__IO uint32_t CRYP_TypeDef::CSGCM7R***
  - CRYP GCM/GMAC context swap register 7, Address offset: 0x8C

9.2 CRYP Firmware driver API description

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

9.2.1 How to use this driver

The CRYP HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
   a. Enable the CRYP interface clock using __CRYP_CLK_ENABLE()
   b. In case of using interrupts (e.g. HAL_CRYP_AESECB_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_AESECB_Encrypt_DMA())
      - Enable the DMAx interface clock using __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 initilalized 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 HAL using HAL_CRYP_Init(). This function configures mainly:
   a. The data type: 1-bit, 8-bit, 16-bit and 32-bit
   b. The key size: 128, 192 and 256. This parameter is relevant only for AES
   c. The encryption/decryption key. It's size depends on the algorithm used for encryption/decryption
   d. The initialization vector (counter). It is not used ECB 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_AESCBC_Encrypt()

b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_AESCBC_Encrypt_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_AESCBC_Encrypt_DMA()

4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is initialized and processes the buffer in input. At second call, the processing function performs an append of the already processed buffer. When a new data block is to be processed, call HAL_CRYP_Init() then the processing function.
5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.

9.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYP according to the specified parameters in the CRYP_InitTypeDef and creates the associated handle
- DeInitialize the CRYP peripheral
- Initialize the CRYP MSP
- DeInitialize CRYP MSP
- ***HAL_CRYP_Init()***
- ***HAL_CRYP_DeInit()***
- ***HAL_CRYP_MspInit()***
- ***HAL_CRYP_MspDeInit()***

9.2.3 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES-128/192/256 using chaining modes
- Decrypt cyphertext using AES-128/192/256 using chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode
- ***HAL_CRYP_AESECB_Encrypt()***
- ***HAL_CRYP_AESCBC_Encrypt()***
- ***HAL_CRYP_AESCTR_Encrypt()***
- ***HAL_CRYP_AESECB_Decrypt()***
- ***HAL_CRYP_AESCBC_Decrypt()***
- ***HAL_CRYP_AESCTR_Decrypt()***
- ***HAL_CRYP_AESECB_Encrypt_IT()***
- ***HAL_CRYP_AESCBC_Encrypt_IT()***
- ***HAL_CRYP_AESCTR_Encrypt_IT()***
- ***HAL_CRYP_AESECB_Decrypt_IT()***
- ***HAL_CRYP_AESCBC_Decrypt_IT()***
- ***HAL_CRYP_AESCTR_Decrypt_IT()***
- ***HAL_CRYP_AESECB_Encrypt_DMA()***
- ***HAL_CRYP_AESCBC_Encrypt_DMA()***
- ***HAL_CRYP_AESCTR_Encrypt_DMA()***
- ***HAL_CRYP_AESECB_Decrypt_DMA()***
- ***HAL_CRYP_AESCBC_Decrypt_DMA()***

- *HAL_CRYP_AESCTR_Decrypt_DMA()*

9.2.4 DES processing functions

This section provides functions allowing to:

- Encrypt plaintext using DES using ECB or CBC chaining modes
- Decrypt cyphertext using ECB or CBC chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode
- *HAL_CRYP_DESECB_Encrypt()*
- *HAL_CRYP_DESECB_Decrypt()*
- *HAL_CRYP_DESCBC_Encrypt()*
- *HAL_CRYP_DESCBC_Decrypt()*
- *HAL_CRYP_DESECB_Encrypt_IT()*
- *HAL_CRYP_DESCBC_Encrypt_IT()*
- *HAL_CRYP_DESECB_Decrypt_IT()*
- *HAL_CRYP_DESCBC_Decrypt_IT()*
- *HAL_CRYP_DESECB_Encrypt_DMA()*
- *HAL_CRYP_DESCBC_Encrypt_DMA()*
- *HAL_CRYP_DESECB_Decrypt_DMA()*
- *HAL_CRYP_DESCBC_Decrypt_DMA()*

9.2.5 TDES processing functions

This section provides functions allowing to:

- Encrypt plaintext using TDES based on ECB or CBC chaining modes
- Decrypt cyphertext using TDES based on ECB or CBC chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode
- *HAL_CRYP_TDESECB_Encrypt()*
- *HAL_CRYP_TDESECB_Decrypt()*
- *HAL_CRYP_TDESCBC_Encrypt()*
- *HAL_CRYP_TDESCBC_Decrypt()*
- *HAL_CRYP_TDESECB_Encrypt_IT()*
- *HAL_CRYP_TDESCBC_Encrypt_IT()*
- *HAL_CRYP_TDESECB_Decrypt_IT()*
- *HAL_CRYP_TDESCBC_Decrypt_IT()*
- *HAL_CRYP_TDESECB_Encrypt_DMA()*
- *HAL_CRYP_TDESCBC_Encrypt_DMA()*
- *HAL_CRYP_TDESECB_Decrypt_DMA()*
- *HAL_CRYP_TDESCBC_Decrypt_DMA()*

9.2.6 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error
- *HAL_CRYP_InCpltCallback()*
- *HAL_CRYP_OutCpltCallback()*
- *HAL_CRYP_ErrorCallback()*

9.2.7 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

- *HAL_CRYP_IRQHandler()*

9.2.8 Peripheral State functions

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

- *HAL_CRYP_GetState()*

9.2.9 Initialization and de-initialization functions

9.2.9.1 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_InitTypeDef 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**
Notes	• None.

9.2.9.2 HAL_CRYP_DeInit

Function Name	**HAL_StatusTypeDef HAL_CRYP_DeInit (** ***CRYP_HandleTypeDef **** **hcryp)**
Function Description	DeInitializes the CRYP peripheral.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values	• **HAL status**
Notes	• None.

9.2.9.3 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.
Notes	• None.

9.2.9.4 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.
Notes	• None.

9.2.10 AES processing functions

9.2.10.1 HAL_CRYP_AESECB_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt (*CRYP_HandleTypeDef * * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**

Function Description	Initializes the CRYP peripheral in AES ECB encryption mode then encrypt pPlainData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.10.2 HAL_CRYP_AESCBC_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode then encrypt pPlainData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.10.3 HAL_CRYP_AESCTR_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode then encrypt pPlainData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

	• **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.10.4 HAL_CRYP_AESECB_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData :** Pointer to the plaintext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.10.5 HAL_CRYP_AESCBC_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData :** Pointer to the plaintext buffer • **Timeout :** Specify Timeout value

Return values	• **HAL status**
Notes	• None.

9.2.10.6 HAL_CRYP_AESCTR_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData :** Pointer to the plaintext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.10.7 HAL_CRYP_AESECB_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using Interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.8 HAL_CRYP_AESCBC_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using Interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.9 HAL_CRYP_AESCTR_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_IT (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using Interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.10 HAL_CRYP_AESECB_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_IT (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using Interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.11 HAL_CRYP_AESCBC_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in AES CBC decryption mode using IT.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.12 HAL_CRYP_AESCTR_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_IT (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData,**

	uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode using Interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.13 HAL_CRYP_AESECB_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 bytes • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.14 HAL_CRYP_AESCBC_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_DMA (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that

	contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.15 HAL_CRYP_AESCTR_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16. • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.16 HAL_CRYP_AESECB_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA (*CRYP_HandleTypeDef ** hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 bytes • **pPlainData :** Pointer to the plaintext buffer

Return values	• **HAL status**
Notes	• None.

9.2.10.17 HAL_CRYP_AESCBC_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 bytes • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.10.18 HAL_CRYP_AESCTR_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 16 • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11 DES processing functions

9.2.11.1 HAL_CRYP_DESECB_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in DES ECB encryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.11.2 HAL_CRYP_DESECB_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.11.3 HAL_CRYP_DESCBC_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in DES CBC encryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.11.4 HAL_CRYP_DESCBC_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.11.5 HAL_CRYP_DESECB_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt_IT (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in DES ECB encryption mode

	using IT.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.6 HAL_CRYP_DESCBC_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt_IT (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in DES CBC encryption mode using interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.7 HAL_CRYP_DESECB_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt_IT (*CRYP_HandleTypeDef ** hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using IT.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8

	• **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.8 HAL_CRYP_DESCBC_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt_IT (** ***CRYP_HandleTypeDef **** **hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.9 HAL_CRYP_DESECB_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt_DMA (** ***CRYP_HandleTypeDef **** **hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in DES ECB encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.10 HAL_CRYP_DESCBC_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in DES CBC encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.11 HAL_CRYP_DESECB_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.11.12 HAL_CRYP_DESCBC_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12 TDES processing functions

9.2.12.1 HAL_CRYP_TDESECB_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt (*CRYP_HandleTypeDef* * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in TDES ECB encryption mode then encrypt pPlainData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.12.2 HAL_CRYP_TDESECB_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData,**

	uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYP peripheral in TDES ECB decryption mode then decrypted pCypherData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.12.3 HAL_CRYP_TDESCBC_Encrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in TDES CBC encryption mode then encrypt pPlainData.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.12.4 HAL_CRYP_TDESCBC_Decrypt

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt (*CRYP_HandleTypeDef ** hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
Function Description	Initializes the CRYP peripheral in TDES CBC decryption mode then decrypted pCypherData.

Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pPlainData :** Pointer to the plaintext buffer • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

9.2.12.5 HAL_CRYP_TDESECB_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt_IT (** ***CRYP_HandleTypeDef ** *** **hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in TDES ECB encryption mode using interrupt.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12.6 HAL_CRYP_TDESCBC_Encrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt_IT (** ***CRYP_HandleTypeDef ** *** **hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in TDES CBC encryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer

Return values	• **HAL status**
Notes	• None.

9.2.12.7 HAL_CRYP_TDESECB_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt_IT (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in TDES ECB decryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12.8 HAL_CRYP_TDESCBC_Decrypt_IT

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt_IT (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in TDES CBC decryption mode.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12.9 HAL_CRYP_TDESECB_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in TDES ECB encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12.10 HAL_CRYP_TDESCBC_Encrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
Function Description	Initializes the CRYP peripheral in TDES CBC encryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12.11 HAL_CRYP_TDESECB_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt_DMA (*CRYP_HandleTypeDef* * hcryp, uint8_t * pCypherData,**

	uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYP peripheral in TDES ECB decryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pPlainData :** Pointer to the plaintext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pCypherData :** Pointer to the cyphertext buffer
Return values	• **HAL status**
Notes	• None.

9.2.12.12 HAL_CRYP_TDESCBC_Decrypt_DMA

Function Name	**HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt_DMA (** ***CRYP_HandleTypeDef** * **hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
Function Description	Initializes the CRYP peripheral in TDES CBC decryption mode using DMA.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • **pCypherData :** Pointer to the cyphertext buffer • **Size :** Length of the plaintext buffer, must be a multiple of 8 • **pPlainData :** Pointer to the plaintext buffer
Return values	• **HAL status**
Notes	• None.

9.2.13 DMA callback functions

9.2.13.1 HAL_CRYP_InCpltCallback

Function Name	**void HAL_CRYP_InCpltCallback (** ***CRYP_HandleTypeDef** * **hcryp)**
Function Description	Input FIFO transfer completed callbacks.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values	• None.
Notes	• None.

9.2.13.2 HAL_CRYP_OutCpltCallback

Function Name	**void HAL_CRYP_OutCpltCallback (*CRYP_HandleTypeDef ** hcryp)**
Function Description	Output FIFO transfer completed callbacks.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• None.
Notes	• None.

9.2.13.3 HAL_CRYP_ErrorCallback

Function Name	**void HAL_CRYP_ErrorCallback (*CRYP_HandleTypeDef ** hcryp)**
Function Description	CRYP error callbacks.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• None.
Notes	• None.

9.2.14 CRYP IRQ handler management

9.2.14.1 HAL_CRYP_IRQHandler

Function Name	**void HAL_CRYP_IRQHandler (*CRYP_HandleTypeDef ** hcryp)**

Function Description	This function handles CRYP interrupt request.
Parameters	• **hcryp :** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	• None.
Notes	• None.

9.2.15 Peripheral State functions

9.2.15.1 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**
Notes	• None.

9.3 CRYP Firmware driver defines

9.3.1 CRYP

CRYP

CRYP_AlgoModeDirection

- #define: ***CRYP_CR_ALGOMODE_DIRECTION ((uint32_t)0x0008003C)***

- #define: ***CRYP_CR_ALGOMODE_TDES_ECB_ENCRYPT ((uint32_t)0x00000000)***

- #define: ***CRYP_CR_ALGOMODE_TDES_ECB_DECRYPT ((uint32_t)0x00000004)***

- #define: ***CRYP_CR_ALGOMODE_TDES_CBC_ENCRYPT ((uint32_t)0x00000008)***

- #define: ***CRYP_CR_ALGOMODE_TDES_CBC_DECRYPT ((uint32_t)0x0000000C)***
- #define: ***CRYP_CR_ALGOMODE_DES_ECB_ENCRYPT ((uint32_t)0x00000010)***
- #define: ***CRYP_CR_ALGOMODE_DES_ECB_DECRYPT ((uint32_t)0x00000014)***
- #define: ***CRYP_CR_ALGOMODE_DES_CBC_ENCRYPT ((uint32_t)0x00000018)***
- #define: ***CRYP_CR_ALGOMODE_DES_CBC_DECRYPT ((uint32_t)0x0000001C)***
- #define: ***CRYP_CR_ALGOMODE_AES_ECB_ENCRYPT ((uint32_t)0x00000020)***
- #define: ***CRYP_CR_ALGOMODE_AES_ECB_DECRYPT ((uint32_t)0x00000024)***
- #define: ***CRYP_CR_ALGOMODE_AES_CBC_ENCRYPT ((uint32_t)0x00000028)***
- #define: ***CRYP_CR_ALGOMODE_AES_CBC_DECRYPT ((uint32_t)0x0000002C)***
- #define: ***CRYP_CR_ALGOMODE_AES_CTR_ENCRYPT ((uint32_t)0x00000030)***
- #define: ***CRYP_CR_ALGOMODE_AES_CTR_DECRYPT ((uint32_t)0x00000034)***

CRYP_Data_Type

- #define: ***CRYP_DATATYPE_32B ((uint32_t)0x00000000)***

- #define: ***CRYP_DATATYPE_16B CRYP_CR_DATATYPE_0***

- #define: ***CRYP_DATATYPE_8B CRYP_CR_DATATYPE_1***

- #define: ***CRYP_DATATYPE_1B CRYP_CR_DATATYPE***

CRYP_Flags

- #define: ***CRYP_FLAG_BUSY ((uint32_t)0x00000010)***

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

- #define: ***CRYP_FLAG_IFEM ((uint32_t)0x00000001)***

Input FIFO is empty

- #define: ***CRYP_FLAG_IFNF ((uint32_t)0x00000002)***

Input FIFO is not Full

- #define: ***CRYP_FLAG_OFNE ((uint32_t)0x00000004)***

Output FIFO is not empty

- #define: ***CRYP_FLAG_OFFU ((uint32_t)0x00000008)***

Output FIFO is Full

- #define: ***CRYP_FLAG_OUTRIS ((uint32_t)0x01000002)***

Output FIFO service raw interrupt status

- #define: ***CRYP_FLAG_INRIS ((uint32_t)0x01000001)***

Input FIFO service raw interrupt status

CRYP_Interrupt

- #define: ***CRYP_IT_INI ((uint32_t)CRYP_IMSCR_INIM)***

Input FIFO Interrupt

- #define: ***CRYP_IT_OUTI ((uint32_t)CRYP_IMSCR_OUTIM)***

Output FIFO Interrupt

CRYP_Key_Size

- #define: ***CRYP_KEYSIZE_128B ((uint32_t)0x00000000)***

- #define: ***CRYP_KEYSIZE_192B CRYP_CR_KEYSIZE_0***

- #define: ***CRYP_KEYSIZE_256B CRYP_CR_KEYSIZE_1***

10 HAL DAC Generic Driver

10.1 DAC Firmware driver registers structures

10.1.1 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

10.1.2 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***

10.1.3 DAC_TypeDef

DAC_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t SWTRIGR***
- ***__IO uint32_t DHR12R1***
- ***__IO uint32_t DHR12L1***
- ***__IO uint32_t DHR8R1***
- ***__IO uint32_t DHR12R2***
- ***__IO uint32_t DHR12L2***
- ***__IO uint32_t DHR8R2***
- ***__IO uint32_t DHR12RD***
- ***__IO uint32_t DHR12LD***
- ***__IO uint32_t DHR8RD***
- ***__IO uint32_t DOR1***
- ***__IO uint32_t DOR2***
- ***__IO uint32_t SR***

Field Documentation

- ***__IO uint32_t DAC_TypeDef::CR***
  - DAC control register, Address offset: 0x00
- ***__IO uint32_t DAC_TypeDef::SWTRIGR***
  - DAC software trigger register, Address offset: 0x04
- ***__IO uint32_t DAC_TypeDef::DHR12R1***
  - DAC channel1 12-bit right-aligned data holding register, Address offset: 0x08
- ***__IO uint32_t DAC_TypeDef::DHR12L1***
  - DAC channel1 12-bit left aligned data holding register, Address offset: 0x0C
- ***__IO uint32_t DAC_TypeDef::DHR8R1***
  - DAC channel1 8-bit right aligned data holding register, Address offset: 0x10
- ***__IO uint32_t DAC_TypeDef::DHR12R2***
  - DAC channel2 12-bit right aligned data holding register, Address offset: 0x14
- ***__IO uint32_t DAC_TypeDef::DHR12L2***
  - DAC channel2 12-bit left aligned data holding register, Address offset: 0x18
- ***__IO uint32_t DAC_TypeDef::DHR8R2***
  - DAC channel2 8-bit right-aligned data holding register, Address offset: 0x1C
- ***__IO uint32_t DAC_TypeDef::DHR12RD***
  - Dual DAC 12-bit right-aligned data holding register, Address offset: 0x20
- ***__IO uint32_t DAC_TypeDef::DHR12LD***
  - DUAL DAC 12-bit left aligned data holding register, Address offset: 0x24
- ***__IO uint32_t DAC_TypeDef::DHR8RD***
  - DUAL DAC 8-bit right aligned data holding register, Address offset: 0x28
- ***__IO uint32_t DAC_TypeDef::DOR1***

- DAC channel1 data output register, Address offset: 0x2C
- ***__IO uint32_t DAC_TypeDef::DOR2***
  - DAC channel2 data output register, Address offset: 0x30
- ***__IO uint32_t DAC_TypeDef::SR***
  - DAC status register, Address offset: 0x34

10.2 DAC Firmware driver API description

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

10.2.1 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;

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

10.2.2 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 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()

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

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

10.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.
- *HAL_DAC_Init()*
- *HAL_DAC_DeInit()*
- *HAL_DAC_MspInit()*
- *HAL_DAC_MspDeInit()*

10.2.4 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.
- *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()*

10.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.
- *HAL_DAC_ConfigChannel()*
- *HAL_DAC_SetValue()*

10.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.
- *HAL_DAC_GetState()*
- *HAL_DAC_GetError()*

10.2.7 Initialization and de-initialization functions

10.2.7.1 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**
Notes	• None.

10.2.7.2 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**
Notes	• None.

10.2.7.3 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.
Notes	• None.

10.2.7.4 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.
Notes	• None.

10.2.8 IO operation functions

10.2.8.1 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**
Notes	• None.

10.2.8.2 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**
Notes	• None.

10.2.8.3 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**
Notes	• None.

10.2.8.4 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**
Notes	• None.

10.2.8.5 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.**
Notes	• None.

10.2.8.6 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.
Notes	• None.

10.2.8.7 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.
Notes	• None.

10.2.8.8 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.

Notes	• None.

10.2.8.9 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.
Notes	• None.

10.2.8.10 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.
Notes	• None.

10.2.9 Peripheral Control functions

10.2.9.1 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**
Notes	• None.

10.2.9.2 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**
Notes	• None.

10.2.10 Peripheral State and Errors functions

10.2.10.1 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**
Notes	• None.

10.2.10.2 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**
Notes	• None.

10.3 DAC Firmware driver defines

10.3.1 DAC

DAC

DAC_Channel_selection

- #define: ***DAC_CHANNEL_1 ((uint32_t)0x00000000)***

- #define: ***DAC_CHANNEL_2 ((uint32_t)0x00000010)***

DAC_data_alignement

- #define: ***DAC_ALIGN_12B_R ((uint32_t)0x00000000)***

- #define: ***DAC_ALIGN_12B_L ((uint32_t)0x00000004)***

- #define: ***DAC_ALIGN_8B_R ((uint32_t)0x00000008)***

DAC_Error_Code

- #define: ***HAL_DAC_ERROR_NONE 0x00***

No error

- #define: ***HAL_DAC_ERROR_DMAUNDERRUNCH1 0x01***

DAC channel1 DAM underrun error

- #define: ***HAL_DAC_ERROR_DMAUNDERRUNCH2 0x02***

DAC channel2 DAM underrun error

- #define: ***HAL_DAC_ERROR_DMA 0x04***

DMA error

DAC_flags_definition

- #define: ***DAC_FLAG_DMAUDR1 ((uint32_t)DAC_SR_DMAUDR1)***

- #define: ***DAC_FLAG_DMAUDR2 ((uint32_t)DAC_SR_DMAUDR2)***

DAC_IT_definition

- #define: ***DAC_IT_DMAUDR1 ((uint32_t)DAC_SR_DMAUDR1)***

- #define: ***DAC_IT_DMAUDR2 ((uint32_t)DAC_SR_DMAUDR2)***

DAC_output_buffer

- #define: ***DAC_OUTPUTBUFFER_ENABLE ((uint32_t)0x00000000)***

- #define: ***DAC_OUTPUTBUFFER_DISABLE ((uint32_t)DAC_CR_BOFF1)***

DAC_trigger_selection

- #define: ***DAC_TRIGGER_NONE ((uint32_t)0x00000000)***

Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger

- #define: ***DAC_TRIGGER_T2_TRGO ((uint32_t)(DAC_CR_TSEL1_2 | DAC_CR_TEN1))***

TIM2 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_T4_TRGO ((uint32_t)(DAC_CR_TSEL1_2 | DAC_CR_TSEL1_0 | DAC_CR_TEN1))***

TIM4 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_T5_TRGO ((uint32_t)(DAC_CR_TSEL1_1 | DAC_CR_TSEL1_0 | DAC_CR_TEN1))***

TIM5 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_T6_TRGO ((uint32_t)DAC_CR_TEN1)***

TIM6 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_T7_TRGO ((uint32_t)(DAC_CR_TSEL1_1 | DAC_CR_TEN1))***

TIM7 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_T8_TRGO ((uint32_t)(DAC_CR_TSEL1_0 | DAC_CR_TEN1))***

TIM8 TRGO selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_EXT_IT9 ((uint32_t)(DAC_CR_TSEL1_2 | DAC_CR_TSEL1_1 | DAC_CR_TEN1))***

EXTI Line9 event selected as external conversion trigger for DAC channel

- #define: ***DAC_TRIGGER_SOFTWARE ((uint32_t)(DAC_CR_TSEL1 | DAC_CR_TEN1))***

Conversion started by software trigger for DAC channel

11 HAL DAC Extension Driver

11.1 DACEx Firmware driver API description

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

11.1.1 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.

11.1.2 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.
- ***HAL_DACEx_DualGetValue()***
- ***HAL_DACEx_TriangleWaveGenerate()***
- ***HAL_DACEx_NoiseWaveGenerate()***
- ***HAL_DACEx_DualSetValue()***

11.1.3 Extended features functions

11.1.3.1 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.**
Notes	• None.

11.1.3.2 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**
Notes	• None.

11.1.3.3 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**
Notes	• None.

11.1.3.4 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.

11.2 DACEx Firmware driver defines

11.2.1 DACEx

DACEx

DACEx_lfsrunmask_triangleamplitude

- #define: ***DAC_LFSRUNMASK_BIT0 ((uint32_t)0x00000000)***

Unmask DAC channel LFSR bit0 for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS1_0 ((uint32_t)DAC_CR_MAMP1_0)***

Unmask DAC channel LFSR bit[1:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS2_0 ((uint32_t)DAC_CR_MAMP1_1)***

Unmask DAC channel LFSR bit[2:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS3_0 ((uint32_t)DAC_CR_MAMP1_1 | DAC_CR_MAMP1_0)***

Unmask DAC channel LFSR bit[3:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS4_0 ((uint32_t)DAC_CR_MAMP1_2)***

Unmask DAC channel LFSR bit[4:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS5_0 ((uint32_t)DAC_CR_MAMP1_2 | DAC_CR_MAMP1_0)***

Unmask DAC channel LFSR bit[5:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS6_0 ((uint32_t)DAC_CR_MAMP1_2 | DAC_CR_MAMP1_1)***

Unmask DAC channel LFSR bit[6:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS7_0 ((uint32_t)DAC_CR_MAMP1_2 | DAC_CR_MAMP1_1 | DAC_CR_MAMP1_0)***

Unmask DAC channel LFSR bit[7:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS8_0 ((uint32_t)DAC_CR_MAMP1_3)***

Unmask DAC channel LFSR bit[8:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS9_0 ((uint32_t)DAC_CR_MAMP1_3 | DAC_CR_MAMP1_0)***

Unmask DAC channel LFSR bit[9:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS10_0 ((uint32_t)DAC_CR_MAMP1_3 | DAC_CR_MAMP1_1)***

Unmask DAC channel LFSR bit[10:0] for noise wave generation

- #define: ***DAC_LFSRUNMASK_BITS11_0 ((uint32_t)DAC_CR_MAMP1_3 | DAC_CR_MAMP1_1 | DAC_CR_MAMP1_0)***

Unmask DAC channel LFSR bit[11:0] for noise wave generation

- #define: ***DAC_TRIANGLEAMPLITUDE_1 ((uint32_t)0x00000000)***

Select max triangle amplitude of 1

- #define: ***DAC_TRIANGLEAMPLITUDE_3 ((uint32_t)DAC_CR_MAMP1_0)***

Select max triangle amplitude of 3

- #define: ***DAC_TRIANGLEAMPLITUDE_7 ((uint32_t)DAC_CR_MAMP1_1)***

Select max triangle amplitude of 7

- #define: ***DAC_TRIANGLEAMPLITUDE_15 ((uint32_t)DAC_CR_MAMP1_1 | DAC_CR_MAMP1_0)***

Select max triangle amplitude of 15

- #define: ***DAC_TRIANGLEAMPLITUDE_31 ((uint32_t)DAC_CR_MAMP1_2)***

Select max triangle amplitude of 31

- #define: ***DAC_TRIANGLEAMPLITUDE_63 ((uint32_t)DAC_CR_MAMP1_2 | DAC_CR_MAMP1_0)***

Select max triangle amplitude of 63

- #define: ***DAC_TRIANGLEAMPLITUDE_127 ((uint32_t)DAC_CR_MAMP1_2 | DAC_CR_MAMP1_1)***

Select max triangle amplitude of 127

- #define: ***DAC_TRIANGLEAMPLITUDE_255 ((uint32_t)DAC_CR_MAMP1_2 | DAC_CR_MAMP1_1 | DAC_CR_MAMP1_0)***

Select max triangle amplitude of 255

- #define: ***DAC_TRIANGLEAMPLITUDE_511 ((uint32_t)DAC_CR_MAMP1_3)***

Select max triangle amplitude of 511

- #define: ***DAC_TRIANGLEAMPLITUDE_1023 ((uint32_t)DAC_CR_MAMP1_3 | DAC_CR_MAMP1_0)***

Select max triangle amplitude of 1023

- #define: ***DAC_TRIANGLEAMPLITUDE_2047 ((uint32_t)DAC_CR_MAMP1_3 | DAC_CR_MAMP1_1)***

Select max triangle amplitude of 2047

- #define: ***DAC_TRIANGLEAMPLITUDE_4095 ((uint32_t)DAC_CR_MAMP1_3 | DAC_CR_MAMP1_1 | DAC_CR_MAMP1_0)***

Select max triangle amplitude of 4095

DACEx_wave_generation

- #define: ***DAC_WAVEGENERATION_NONE ((uint32_t)0x00000000)***

- #define: ***DAC_WAVEGENERATION_NOISE ((uint32_t)DAC_CR_WAVE1_0)***

- #define: ***DAC_WAVEGENERATION_TRIANGLE ((uint32_t)DAC_CR_WAVE1_1)***

- #define: ***DAC_WAVE_NOISE ((uint32_t)DAC_CR_WAVE1_0)***

- #define: ***DAC_WAVE_TRIANGLE ((uint32_t)DAC_CR_WAVE1_1)***

12 HAL DMA Generic Driver

12.1 DMA Firmware driver registers structures

12.1.1 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(* XferErrorCallback***
- ***__IO uint32_t ErrorCode***

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::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  - DMA transfer error callback
- ***__IO uint32_t DMA_HandleTypeDef::ErrorCode***
  - DMA Error code

12.1.2 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***
- ***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***
- ***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 interruptable transaction. This parameter can be a value of ***DMA_Memory_burst***
- ***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 interruptable transaction. This parameter can be a value of ***DMA_Peripheral_burst***

12.1.3 DMA_Stream_TypeDef

DMA_Stream_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t NDTR***
- ***__IO uint32_t PAR***
- ***__IO uint32_t M0AR***
- ***__IO uint32_t M1AR***
- ***__IO uint32_t FCR***

Field Documentation

- ***__IO uint32_t DMA_Stream_TypeDef::CR***
  - DMA stream x configuration register
- ***__IO uint32_t DMA_Stream_TypeDef::NDTR***
  - DMA stream x number of data register
- ***__IO uint32_t DMA_Stream_TypeDef::PAR***
  - DMA stream x peripheral address register
- ***__IO uint32_t DMA_Stream_TypeDef::M0AR***
  - DMA stream x memory 0 address register
- ***__IO uint32_t DMA_Stream_TypeDef::M1AR***
  - DMA stream x memory 1 address register
- ***__IO uint32_t DMA_Stream_TypeDef::FCR***
  - DMA stream x FIFO control register

12.1.4 DMA_TypeDef

DMA_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t LISR***
- ***__IO uint32_t HISR***
- ***__IO uint32_t LIFCR***
- ***__IO uint32_t HIFCR***

Field Documentation

- ***__IO uint32_t DMA_TypeDef::LISR***
  - DMA low interrupt status register, Address offset: 0x00
- ***__IO uint32_t DMA_TypeDef::HISR***
  - DMA high interrupt status register, Address offset: 0x04
- ***__IO uint32_t DMA_TypeDef::LIFCR***
  - DMA low interrupt flag clear register, Address offset: 0x08
- ***__IO uint32_t DMA_TypeDef::HIFCR***
  - DMA high interrupt flag clear register, Address offset: 0x0C

12.2 DMA Firmware driver API description

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

12.2.1 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.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function 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() function to abort the current transfer In Memory-to-Memory transfer mode, Circular mode is not allowed. 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). 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_FS: Return the current DMA Stream FIFO filled level.
- __HAL_DMA_GET_FLAG: Get the DMA Stream pending flags.
- __HAL_DMA_CLEAR_FLAG: Clear the DMA Stream pending flags.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Stream interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Stream interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Stream interrupt has occurred or not.

You can refer to the DMA HAL driver header file for more useful macros

12.2.2 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.

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

12.2.3 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
- ***HAL_DMA_Start()***

- *HAL_DMA_Start_IT()*
- *HAL_DMA_Abort()*
- *HAL_DMA_PollForTransfer()*
- *HAL_DMA_IRQHandler()*

12.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code
- *HAL_DMA_GetState()*
- *HAL_DMA_GetError()*

12.2.5 Initialization and de-initialization functions

12.2.5.1 HAL_DMA_Init

Function Name	**HAL_StatusTypeDef HAL_DMA_Init (*DMA_HandleTypeDef ** hdma)**
Function Description	Initializes 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**
Notes	• None.

12.2.5.2 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**
Notes	• None.

12.2.6 I/O operation functions

12.2.6.1 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**
Notes	• None.

12.2.6.2 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**
Notes	• None.

12.2.6.3 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.

12.2.6.4 HAL_DMA_PollForTransfer

Function Name	**HAL_StatusTypeDef HAL_DMA_PollForTransfer (*DMA_HandleTypeDef* * hdma, uint32_t 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	• None.

12.2.6.5 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.
Notes	• None.

12.2.7 Peripheral State functions

12.2.7.1 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**
Notes	• None.

12.2.7.2 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**
Notes	• None.

12.3 DMA Firmware driver defines

12.3.1 DMA

DMA

DMA_Channel_selection

- #define: ***DMA_CHANNEL_0 ((uint32_t)0x00000000)***

DMA Channel 0

- #define: ***DMA_CHANNEL_1 ((uint32_t)0x02000000)***

DMA Channel 1

- #define: ***DMA_CHANNEL_2 ((uint32_t)0x04000000)***

DMA Channel 2

- #define: ***DMA_CHANNEL_3 ((uint32_t)0x06000000)***

DMA Channel 3

- #define: ***DMA_CHANNEL_4 ((uint32_t)0x08000000)***

DMA Channel 4

- #define: ***DMA_CHANNEL_5 ((uint32_t)0x0A000000)***

DMA Channel 5

- #define: ***DMA_CHANNEL_6 ((uint32_t)0x0C000000)***

DMA Channel 6

- #define: ***DMA_CHANNEL_7 ((uint32_t)0x0E000000)***

DMA Channel 7

DMA_Data_transfer_direction

- #define: ***DMA_PERIPH_TO_MEMORY ((uint32_t)0x00000000)***

Peripheral to memory direction

- #define: ***DMA_MEMORY_TO_PERIPH ((uint32_t)DMA_SxCR_DIR_0)***

Memory to peripheral direction

- #define: ***DMA_MEMORY_TO_MEMORY ((uint32_t)DMA_SxCR_DIR_1)***

Memory to memory direction

DMA_Error_Code

- #define: ***HAL_DMA_ERROR_NONE ((uint32_t)0x00000000)***

No error

- #define: ***HAL_DMA_ERROR_TE ((uint32_t)0x00000001)***

Transfer error

- #define: ***HAL_DMA_ERROR_FE ((uint32_t)0x00000002)***

FIFO error

- #define: ***HAL_DMA_ERROR_DME ((uint32_t)0x00000004)***

Direct Mode error

- #define: ***HAL_DMA_ERROR_TIMEOUT ((uint32_t)0x00000020)***

Timeout error

DMA_FIFO_direct_mode

- #define: ***DMA_FIFOMODE_DISABLE ((uint32_t)0x00000000)***

FIFO mode disable

- #define: ***DMA_FIFOMODE_ENABLE ((uint32_t)DMA_SxFCR_DMDIS)***

FIFO mode enable

DMA_FIFO_threshold_level

- #define: ***DMA_FIFO_THRESHOLD_1QUARTERFULL ((uint32_t)0x00000000)***

FIFO threshold 1 quart full configuration

- #define: ***DMA_FIFO_THRESHOLD_HALFFULL ((uint32_t)DMA_SxFCR_FTH_0)***

FIFO threshold half full configuration

- #define: ***DMA_FIFO_THRESHOLD_3QUARTERSFULL ((uint32_t)DMA_SxFCR_FTH_1)***

FIFO threshold 3 quarts full configuration

- #define: ***DMA_FIFO_THRESHOLD_FULL ((uint32_t)DMA_SxFCR_FTH)***

FIFO threshold full configuration

DMA_flag_definitions

- #define: ***DMA_FLAG_FEIF0_4 ((uint32_t)0x00800001)***
- #define: ***DMA_FLAG_DMEIF0_4 ((uint32_t)0x00800004)***
- #define: ***DMA_FLAG_TEIF0_4 ((uint32_t)0x00000008)***
- #define: ***DMA_FLAG_HTIF0_4 ((uint32_t)0x00000010)***
- #define: ***DMA_FLAG_TCIF0_4 ((uint32_t)0x00000020)***
- #define: ***DMA_FLAG_FEIF1_5 ((uint32_t)0x00000040)***
- #define: ***DMA_FLAG_DMEIF1_5 ((uint32_t)0x00000100)***
- #define: ***DMA_FLAG_TEIF1_5 ((uint32_t)0x00000200)***
- #define: ***DMA_FLAG_HTIF1_5 ((uint32_t)0x00000400)***
- #define: ***DMA_FLAG_TCIF1_5 ((uint32_t)0x00000800)***
- #define: ***DMA_FLAG_FEIF2_6 ((uint32_t)0x00010000)***

- #define: ***DMA_FLAG_DMEIF2_6 ((uint32_t)0x00040000)***

- #define: ***DMA_FLAG_TEIF2_6 ((uint32_t)0x00080000)***

- #define: ***DMA_FLAG_HTIF2_6 ((uint32_t)0x00100000)***

- #define: ***DMA_FLAG_TCIF2_6 ((uint32_t)0x00200000)***

- #define: ***DMA_FLAG_FEIF3_7 ((uint32_t)0x00400000)***

- #define: ***DMA_FLAG_DMEIF3_7 ((uint32_t)0x01000000)***

- #define: ***DMA_FLAG_TEIF3_7 ((uint32_t)0x02000000)***

- #define: ***DMA_FLAG_HTIF3_7 ((uint32_t)0x04000000)***

- #define: ***DMA_FLAG_TCIF3_7 ((uint32_t)0x08000000)***

DMA_Handle_index

- #define: ***TIM_DMA_ID_UPDATE ((uint16_t) 0x0)***

Index of the DMA handle used for Update DMA requests

- #define: ***TIM_DMA_ID_CC1 ((uint16_t) 0x1)***

Index of the DMA handle used for Capture/Compare 1 DMA requests

- #define: ***TIM_DMA_ID_CC2 ((uint16_t) 0x2)***

Index of the DMA handle used for Capture/Compare 2 DMA requests

- #define: ***TIM_DMA_ID_CC3 ((uint16_t) 0x3)***

Index of the DMA handle used for Capture/Compare 3 DMA requests

- #define: ***TIM_DMA_ID_CC4 ((uint16_t) 0x4)***

Index of the DMA handle used for Capture/Compare 4 DMA requests

- #define: ***TIM_DMA_ID_COMMUTATION ((uint16_t) 0x5)***

Index of the DMA handle used for Commutation DMA requests

- #define: ***TIM_DMA_ID_TRIGGER ((uint16_t) 0x6)***

Index of the DMA handle used for Trigger DMA requests

DMA_interrupt_enable_definitions

- #define: ***DMA_IT_TC ((uint32_t)DMA_SxCR_TCIE)***

- #define: ***DMA_IT_HT ((uint32_t)DMA_SxCR_HTIE)***

- #define: ***DMA_IT_TE ((uint32_t)DMA_SxCR_TEIE)***

- #define: ***DMA_IT_DME ((uint32_t)DMA_SxCR_DMEIE)***

- #define: ***DMA_IT_FE ((uint32_t)0x00000080)***

DMA_Memory_burst

- #define: ***DMA_MBURST_SINGLE ((uint32_t)0x00000000)***

- #define: ***DMA_MBURST_INC4 ((uint32_t)DMA_SxCR_MBURST_0)***

- #define: ***DMA_MBURST_INC8 ((uint32_t)DMA_SxCR_MBURST_1)***

- #define: ***DMA_MBURST_INC16 ((uint32_t)DMA_SxCR_MBURST)***

DMA_Memory_data_size

- #define: ***DMA_MDATAALIGN_BYTE ((uint32_t)0x00000000)***

Memory data alignment: Byte

- #define: ***DMA_MDATAALIGN_HALFWORD ((uint32_t)DMA_SxCR_MSIZE_0)***

Memory data alignment: HalfWord

- #define: ***DMA_MDATAALIGN_WORD ((uint32_t)DMA_SxCR_MSIZE_1)***

Memory data alignment: Word

DMA_Memory_incremented_mode

- #define: ***DMA_MINC_ENABLE ((uint32_t)DMA_SxCR_MINC)***

Memory increment mode enable

- #define: ***DMA_MINC_DISABLE ((uint32_t)0x00000000)***

Memory increment mode disable

DMA_mode

- #define: ***DMA_NORMAL ((uint32_t)0x00000000)***

Normal mode

- #define: ***DMA_CIRCULAR ((uint32_t)DMA_SxCR_CIRC)***

Circular mode

- #define: ***DMA_PFCTRL ((uint32_t)DMA_SxCR_PFCTRL)***

Peripheral flow control mode

DMA_Peripheral_burst

- #define: ***DMA_PBURST_SINGLE ((uint32_t)0x00000000)***

- #define: ***DMA_PBURST_INC4 ((uint32_t)DMA_SxCR_PBURST_0)***

- #define: ***DMA_PBURST_INC8 ((uint32_t)DMA_SxCR_PBURST_1)***

- #define: ***DMA_PBURST_INC16 ((uint32_t)DMA_SxCR_PBURST)***

DMA_Peripheral_data_size

- #define: ***DMA_PDATAALIGN_BYTE ((uint32_t)0x00000000)***

Peripheral data alignment: Byte

- #define: ***DMA_PDATAALIGN_HALFWORD ((uint32_t)DMA_SxCR_PSIZE_0)***

Peripheral data alignment: HalfWord

- #define: ***DMA_PDATAALIGN_WORD ((uint32_t)DMA_SxCR_PSIZE_1)***

Peripheral data alignment: Word

DMA_Peripheral_incremented_mode

- #define: ***DMA_PINC_ENABLE ((uint32_t)DMA_SxCR_PINC)***

Peripheral increment mode enable

- #define: ***DMA_PINC_DISABLE ((uint32_t)0x00000000)***

Peripheral increment mode disable

DMA_Priority_level

- #define: ***DMA_PRIORITY_LOW ((uint32_t)0x00000000)***

Priority level: Low

- #define: ***DMA_PRIORITY_MEDIUM ((uint32_t)DMA_SxCR_PL_0)***

Priority level: Medium

- #define: ***DMA_PRIORITY_HIGH ((uint32_t)DMA_SxCR_PL_1)***

Priority level: High

- #define: ***DMA_PRIORITY_VERY_HIGH ((uint32_t)DMA_SxCR_PL)***

Priority level: Very High

13 HAL DMA Extension Driver

13.1 DMAEx Firmware driver API description

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

13.1.1 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. In Memory-to-Memory transfer mode, Multi (Double) Buffer mode is not allowed. When Multi (Double) Buffer mode is enabled the, transfer is circular by default. 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.

13.1.2 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.
- ***HAL_DMAEx_MultiBufferStart()***
- ***HAL_DMAEx_MultiBufferStart_IT()***
- ***HAL_DMAEx_ChangeMemory()***

13.1.3 Extended features functions

13.1.3.1 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**
Notes	• None.

13.1.3.2 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**
Notes	• None.

13.1.3.3 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.

13.2 DMAEx Firmware driver defines

13.2.1 DMAEx

DMAEx

14 HAL DMA2D Generic Driver

14.1 DMA2D Firmware driver registers structures

14.1.1 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

14.1.2 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_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.

14.1.3 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 offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputColorMode***
  – configures the DMA2D foreground color mode . This parameter can be one value of ***DMA2D_Input_Color_Mode***
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaMode***
  – configures the DMA2D foreground alpha mode. This parameter can be one value of ***DMA2D_ALPHA_MODE***
- ***uint32_t DMA2D_LayerCfgTypeDef::InputAlpha***
  – Specifies the DMA2D foreground alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

14.1.4 DMA2D_ColorTypeDef

DMA2D_ColorTypeDef is defined in the stm32f4xx_hal_dma2d.h

Data Fields

- ***uint32_t Blue***
- ***uint32_t Green***
- ***uint32_t Red***

Field Documentation

- ***uint32_t DMA2D_ColorTypeDef::Blue***
  - Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t DMA2D_ColorTypeDef::Green***
  - Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t DMA2D_ColorTypeDef::Red***
  - Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

14.1.5 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.

14.1.6 DMA2D_TypeDef

DMA2D_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t ISR***
- ***__IO uint32_t IFCR***
- ***__IO uint32_t FGMAR***

- ***__IO uint32_t FGOR***
- ***__IO uint32_t BGMAR***
- ***__IO uint32_t BGOR***
- ***__IO uint32_t FGPFCCR***
- ***__IO uint32_t FGCOLR***
- ***__IO uint32_t BGPFCCR***
- ***__IO uint32_t BGCOLR***
- ***__IO uint32_t FGCMAR***
- ***__IO uint32_t BGCMAR***
- ***__IO uint32_t OPFCCR***
- ***__IO uint32_t OCOLR***
- ***__IO uint32_t OMAR***
- ***__IO uint32_t OOR***
- ***__IO uint32_t NLR***
- ***__IO uint32_t LWR***
- ***__IO uint32_t AMTCR***
- ***uint32_t RESERVED***
- ***__IO uint32_t FGCLUT***
- ***__IO uint32_t BGCLUT***

Field Documentation

- ***__IO uint32_t DMA2D_TypeDef::CR***
  - DMA2D Control Register, Address offset: 0x00
- ***__IO uint32_t DMA2D_TypeDef::ISR***
  - DMA2D Interrupt Status Register, Address offset: 0x04
- ***__IO uint32_t DMA2D_TypeDef::IFCR***
  - DMA2D Interrupt Flag Clear Register, Address offset: 0x08
- ***__IO uint32_t DMA2D_TypeDef::FGMAR***
  - DMA2D Foreground Memory Address Register, Address offset: 0x0C
- ***__IO uint32_t DMA2D_TypeDef::FGOR***
  - DMA2D Foreground Offset Register, Address offset: 0x10
- ***__IO uint32_t DMA2D_TypeDef::BGMAR***
  - DMA2D Background Memory Address Register, Address offset: 0x14
- ***__IO uint32_t DMA2D_TypeDef::BGOR***
  - DMA2D Background Offset Register, Address offset: 0x18
- ***__IO uint32_t DMA2D_TypeDef::FGPFCCR***
  - DMA2D Foreground PFC Control Register, Address offset: 0x1C
- ***__IO uint32_t DMA2D_TypeDef::FGCOLR***
  - DMA2D Foreground Color Register, Address offset: 0x20
- ***__IO uint32_t DMA2D_TypeDef::BGPFCCR***
  - DMA2D Background PFC Control Register, Address offset: 0x24
- ***__IO uint32_t DMA2D_TypeDef::BGCOLR***
  - DMA2D Background Color Register, Address offset: 0x28
- ***__IO uint32_t DMA2D_TypeDef::FGCMAR***
  - DMA2D Foreground CLUT Memory Address Register, Address offset: 0x2C
- ***__IO uint32_t DMA2D_TypeDef::BGCMAR***
  - DMA2D Background CLUT Memory Address Register, Address offset: 0x30
- ***__IO uint32_t DMA2D_TypeDef::OPFCCR***
  - DMA2D Output PFC Control Register, Address offset: 0x34
- ***__IO uint32_t DMA2D_TypeDef::OCOLR***

  - DMA2D Output Color Register, Address offset: 0x38
- ***__IO uint32_t DMA2D_TypeDef::OMAR***
  - DMA2D Output Memory Address Register, Address offset: 0x3C
- ***__IO uint32_t DMA2D_TypeDef::OOR***
  - DMA2D Output Offset Register, Address offset: 0x40
- ***__IO uint32_t DMA2D_TypeDef::NLR***
  - DMA2D Number of Line Register, Address offset: 0x44
- ***__IO uint32_t DMA2D_TypeDef::LWR***
  - DMA2D Line Watermark Register, Address offset: 0x48
- ***__IO uint32_t DMA2D_TypeDef::AMTCR***
  - DMA2D AHB Master Timer Configuration Register, Address offset: 0x4C
- ***uint32_t DMA2D_TypeDef::RESERVED[236]***
  - Reserved, 0x50-0x3FF
- ***__IO uint32_t DMA2D_TypeDef::FGCLUT[256]***
  - DMA2D Foreground CLUT, Address offset:400-7FF
- ***__IO uint32_t DMA2D_TypeDef::BGCLUT[256]***
  - DMA2D Background CLUT, Address offset:800-BFF

14.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

14.2.1 How to use this driver

1. Program the required configuration through following parameters: the Transfer Mode, the output color mode and the output offset using HAL_DMA2D_Init() function.
2. Program the required configuration through following parameters: the input color mode, the input color, input alpha value, alpha mode and the input offset using HAL_DMA2D_ConfigLayer() function for foreground or/and background layer.

Polling mode IO operation

- Configure the pdata, Destination and data length and Enable the transfer using HAL_DMA2D_Start()
- 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 the pdata, 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 and XferErrorCallback (i.e a member of DMA2D handle structure). In Register-to-Memory transfer mode, the pdata parameter is the register color, in Memory-to-memory or memory-to-memory with pixel format conversion the pdata is the source address and

it is the color value for the A4 or A8 mode. Configure the foreground source address, the background source address, the Destination and data length and Enable the transfer using HAL_DMA2D_BlendingStart() in polling mode and HAL_DMA2D_BlendingStart_IT() in interrupt mode. HAL_DMA2D_BlendingStart() and HAL_DMA2D_BlendingStart_IT() functions are used if the memory to memory with blending transfer mode is selected.

4. Optionally, configure and enable the CLUT using HAL_DMA2D_ConfigCLUT() HAL_DMA2D_EnableCLUT() functions.
5. Optionally, configure and enable LineInterrupt using the following function: HAL_DMA2D_ProgramLineEvent().
6. The transfer can be suspended, continued and aborted using the following functions: HAL_DMA2D_Suspend(), HAL_DMA2D_Resume(), HAL_DMA2D_Abort().
7. To control DMA2D state you can use the following function: HAL_DMA2D_GetState()

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_DISABLE: Disable 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 has occurred or not.

You can refer to the DMA2D HAL driver header file for more useful macros

14.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D
- ***HAL_DMA2D_Init()***
- ***HAL_DMA2D_DeInit()***
- ***HAL_DMA2D_MspInit()***
- ***HAL_DMA2D_MspDeInit()***

14.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size and Start DMA2D transfer.
- Configure the source for foreground and background, destination address and data size and Start MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size and Start DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size and Start MultiBuffer DMA2D transfer with interrupt.

- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Continue DMA2D transfer.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- *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_PollForTransfer()*
- *HAL_DMA2D_IRQHandler()*

14.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or/and background parameters.
- Configure the DMA2D CLUT transfer.
- Enable DMA2D CLUT.
- Disable DMA2D CLUT.
- Configure the line watermark
- *HAL_DMA2D_ConfigLayer()*
- *HAL_DMA2D_ConfigCLUT()*
- *HAL_DMA2D_EnableCLUT()*
- *HAL_DMA2D_DisableCLUT()*
- *HAL_DMA2D_ProgramLineEvent()*

14.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to :

- Check the DMA2D state
- Get error code
- *HAL_DMA2D_GetState()*
- *HAL_DMA2D_GetError()*

14.2.6 Initialization and Configuration functions

14.2.6.1 HAL_DMA2D_Init

Function Name	**HAL_StatusTypeDef HAL_DMA2D_Init (*DMA2D_HandleTypeDef ** hdma2d)**
Function Description	Initializes 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**
Notes	• None.

14.2.6.2 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.
Notes	• None.

14.2.6.3 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.
Notes	• None.

14.2.6.4 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.
Notes	• None.

14.2.7 IO operation functions

14.2.7.1 HAL_DMA2D_Start

Function Name	**HAL_StatusTypeDef HAL_DMA2D_Start (*DMA2D_HandleTypeDef ** hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Heigh)**
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 the memory to memory or memory to memory with pixel format conversion DMA2D mode is selected, and configure the color value if register to memory DMA2D mode is selected or the color value for the A4 or A8 mode. • **DstAddress :** The destination memory Buffer address. • **Width :** The width of data to be transferred from source to destination. • **Heigh :** The heigh of data to be transferred from source to destination.
Return values	• **HAL status**
Notes	• None.

14.2.7.2 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 Heigh)**

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 DMA2D mode is selected, and configure the color value if register to memory DMA2D mode is selected or the color value for the A4 or A8 mode. • **DstAddress :** The destination memory Buffer address. • **Width :** The width of data to be transferred from source to destination. • **Heigh :** The heigh of data to be transferred from source to destination.
Return values	• **HAL status**
Notes	• None.

14.2.7.3 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 Heigh)**
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 of the foreground layer. • **SrcAddress2 :** The source memory Buffer address of the background layer or the color value for the A4 or A8 mode. • **DstAddress :** The destination memory Buffer address • **Width :** The width of data to be transferred from source to destination. • **Heigh :** The heigh of data to be transferred from source to destination.
Return values	• **HAL status**
Notes	• None.

14.2.7.4 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 Heigh)**
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 of the foreground layer. • **SrcAddress2 :** The source memory Buffer address of the background layer or the color value for the A4 or A8 mode. • **DstAddress :** The destination memory Buffer address. • **Width :** The width of data to be transferred from source to destination. • **Heigh :** The heigh of data to be transferred from source to destination.
Return values	• **HAL status**
Notes	• None.

14.2.7.5 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**
Notes	• None.

14.2.7.6 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**
Notes	• None.

14.2.7.7 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**
Notes	• None.

14.2.7.8 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**
Notes	• None.

14.2.7.9 HAL_DMA2D_IRQHandler

Function Name	**void HAL_DMA2D_IRQHandler (*DMA2D_HandleTypeDef* * hdma2d)**
Function Description	Handles DMA2D interrupt request.
Parameters	• **hdma2d :** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	• **HAL status**
Notes	• None.

14.2.8 Peripheral Control functions

14.2.8.1 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_InitTypeDef and create the associated handle.
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: 0(background) / 1(foreground)
Return values	• **HAL status**
Notes	• None.

14.2.8.2 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: 0(background) / 1(foreground)
Return values	• **HAL status**
Notes	• None.

14.2.8.3 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: 0(background) / 1(foreground)
Return values	• **HAL status**
Notes	• None.

14.2.8.4 HAL_DMA2D_DisableCLUT

Function Name	**HAL_StatusTypeDef HAL_DMA2D_DisableCLUT (*DMA2D_HandleTypeDef ** hdma2d, uint32_t LayerIdx)**
Function Description	Disable 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: 0(background) / 1(foreground)
Return values	• **HAL status**
Notes	• None.

14.2.8.5 HAL_DMA2D_ProgramLineEvent

Function Name	**HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (*DMA2D_HandleTypeDef* * hdma2d, uint32_t Line)**
Function Description	Define the configuration of the line watermark .
Parameters	• **hdma2d :** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D. • **Line :** Line Watermark configuration.
Return values	• **HAL status**
Notes	• None.

14.2.9 Peripheral State functions

14.2.9.1 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**
Notes	• None.

14.2.9.2 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**

Notes • None.

14.3 DMA2D Firmware driver defines

14.3.1 DMA2D

DMA2D

DMA2D_ALPHA_MODE

- #define: ***DMA2D_NO_MODIF_ALPHA ((uint32_t)0x00000000)***

No modification of the alpha channel value

- #define: DMA2D ***DMA2D_REPLACE_ALPHA ((uint32_t)0x00000001)***

Replace original alpha channel value by programmed alpha value

- #define: ***DMA2D_COMBINE_ALPHA ((uint32_t)0x00000002)***

Replace original alpha channel value by programmed alpha value with original alpha channel value

DMA2D_CLUT_CM

- #define: ***DMA2D_CCM_ARGB8888 ((uint32_t)0x00000000)***

ARGB8888 DMA2D C-LUT color mode

- #define: ***DMA2D_CCM_RGB888 ((uint32_t)0x00000001)***

RGB888 DMA2D C-LUT color mode

DMA2D_Clut_Size

- #define: ***DMA2D_CLUT_SIZE (DMA2D_FGPFCCR_CS >> 8)***

DMA2D C-LUT size

DMA2D_Color_Mode

- #define: ***DMA2D_ARGB8888 ((uint32_t)0x00000000)***

ARGB8888 DMA2D color mode

- #define: ***DMA2D_RGB888 ((uint32_t)0x00000001)***

RGB888 DMA2D color mode

- #define: ***DMA2D_RGB565 ((uint32_t)0x00000002)***

RGB565 DMA2D color mode

- #define: ***DMA2D_ARGB1555 ((uint32_t)0x00000003)***

ARGB1555 DMA2D color mode

- #define: ***DMA2D_ARGB4444 ((uint32_t)0x00000004)***

ARGB4444 DMA2D color mode

DMA2D_COLOR_VALUE

- #define: ***COLOR_VALUE ((uint32_t)0x000000FF)***

color value mask

DMA2D_DeadTime

- #define: ***LINE_WATERMARK DMA2D_LWR_LW***

DMA2D_Error_Code

- #define: ***HAL_DMA2D_ERROR_NONE ((uint32_t)0x00000000)***

No error

- #define: ***HAL_DMA2D_ERROR_TE ((uint32_t)0x00000001)***

Transfer error

- #define: ***HAL_DMA2D_ERROR_CE ((uint32_t)0x00000002)***

Configuration error

- #define: ***HAL_DMA2D_ERROR_TIMEOUT ((uint32_t)0x00000020)***

Timeout error

DMA2D_Flag

- #define: ***DMA2D_FLAG_CE DMA2D_ISR_CEIF***

Configuration Error Interrupt Flag

- #define: ***DMA2D_FLAG_CTC DMA2D_ISR_CTCIF***

C-LUT Transfer Complete Interrupt Flag

- #define: ***DMA2D_FLAG_CAE DMA2D_ISR_CAEIF***

C-LUT Access Error Interrupt Flag

- #define: ***DMA2D_FLAG_TW DMA2D_ISR_TWIF***

Transfer Watermark Interrupt Flag

- #define: ***DMA2D_FLAG_TC DMA2D_ISR_TCIF***

Transfer Complete Interrupt Flag

- #define: ***DMA2D_FLAG_TE DMA2D_ISR_TEIF***

Transfer Error Interrupt Flag

DMA2D_Input_Color_Mode

- #define: ***CM_ARGB8888 ((uint32_t)0x00000000)***

ARGB8888 color mode

- #define: ***CM_RGB888 ((uint32_t)0x00000001)***

RGB888 color mode

- #define: ***CM_RGB565 ((uint32_t)0x00000002)***

RGB565 color mode

- #define: ***CM_ARGB1555 ((uint32_t)0x00000003)***

ARGB1555 color mode

- #define: ***CM_ARGB4444 ((uint32_t)0x00000004)***

ARGB4444 color mode

- #define: ***CM_L8 ((uint32_t)0x00000005)***

L8 color mode

- #define: ***CM_AL44 ((uint32_t)0x00000006)***

AL44 color mode

- #define: ***CM_AL88 ((uint32_t)0x00000007)***

AL88 color mode

- #define: ***CM_L4 ((uint32_t)0x00000008)***

L4 color mode

- #define: ***CM_A8 ((uint32_t)0x00000009)***

A8 color mode

- #define: ***CM_A4 ((uint32_t)0x0000000A)***

A4 color mode

DMA2D_Interrupts

- #define: ***DMA2D_IT_CE DMA2D_CR_CEIE***

Configuration Error Interrupt

- #define: ***DMA2D_IT_CTC DMA2D_CR_CTCIE***

C-LUT Transfer Complete Interrupt

- #define: ***DMA2D_IT_CAE DMA2D_CR_CAEIE***

C-LUT Access Error Interrupt

- #define: ***DMA2D_IT_TW DMA2D_CR_TWIE***

Transfer Watermark Interrupt

- #define: ***DMA2D_IT_TC DMA2D_CR_TCIE***

Transfer Complete Interrupt

- #define: ***DMA2D_IT_TE DMA2D_CR_TEIE***

Transfer Error Interrupt

DMA2D_Mode

- #define: ***DMA2D_M2M ((uint32_t)0x00000000)***

DMA2D memory to memory transfer mode

- #define: ***DMA2D_M2M_PFC ((uint32_t)0x00010000)***

DMA2D memory to memory with pixel format conversion transfer mode

- #define: ***DMA2D_M2M_BLEND ((uint32_t)0x00020000)***

DMA2D memory to memory with blending transfer mode

- #define: ***DMA2D_R2M ((uint32_t)0x00030000)***

DMA2D register to memory transfer mode

DMA2D_Offset

- #define: ***DMA2D_OFFSET DMA2D_FGOR_LO***

Line Offset

DMA2D_SIZE

- #define: ***DMA2D_PIXEL (DMA2D_NLR_PL >> 16)***

DMA2D pixel per line

- #define: ***DMA2D_LINE DMA2D_NLR_NL***

DMA2D number of line

15 HAL DCMI Generic Driver

15.1 DCMI Firmware driver registers structures

15.1.1 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

15.1.2 DCMI_InitTypeDef

DCMI_InitTypeDef is defined in the stm32f4xx_hal_dcmi.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***

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***

15.1.3 DCMI_CodesInitTypeDef

DCMI_CodesInitTypeDef is defined in the stm32f4xx_hal_dcmi.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.

15.1.4 DCMI_TypeDef

DCMI_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t SR***
- ***__IO uint32_t RISR***
- ***__IO uint32_t IER***
- ***__IO uint32_t MISR***
- ***__IO uint32_t ICR***
- ***__IO uint32_t ESCR***
- ***__IO uint32_t ESUR***
- ***__IO uint32_t CWSTRTR***
- ***__IO uint32_t CWSIZER***
- ***__IO uint32_t DR***

Field Documentation

- ***__IO uint32_t DCMI_TypeDef::CR***
  – DCMI control register 1, Address offset: 0x00
- ***__IO uint32_t DCMI_TypeDef::SR***
  – DCMI status register, Address offset: 0x04
- ***__IO uint32_t DCMI_TypeDef::RISR***
  – DCMI raw interrupt status register, Address offset: 0x08
- ***__IO uint32_t DCMI_TypeDef::IER***
  – DCMI interrupt enable register, Address offset: 0x0C
- ***__IO uint32_t DCMI_TypeDef::MISR***
  – DCMI masked interrupt status register, Address offset: 0x10
- ***__IO uint32_t DCMI_TypeDef::ICR***
  – DCMI interrupt clear register, Address offset: 0x14
- ***__IO uint32_t DCMI_TypeDef::ESCR***
  – DCMI embedded synchronization code register, Address offset: 0x18
- ***__IO uint32_t DCMI_TypeDef::ESUR***
  – DCMI embedded synchronization unmask register, Address offset: 0x1C
- ***__IO uint32_t DCMI_TypeDef::CWSTRTR***
  – DCMI crop window start, Address offset: 0x20

- ***__IO uint32_t DCMI_TypeDef::CWSIZER***
  – DCMI crop window size, Address offset: 0x24
- ***__IO uint32_t DCMI_TypeDef::DR***
  – DCMI data register, Address offset: 0x28

15.2 DCMI Firmware driver API description

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

15.2.1 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.

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

15.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI
- ***HAL_DCMI_Init()***
- ***HAL_DCMI_DeInit()***
- ***HAL_DCMI_MspInit()***
- ***HAL_DCMI_MspDeInit()***

15.2.3 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.
- handle DCMI interrupt request.
- ***HAL_DCMI_Start_DMA()***
- ***HAL_DCMI_Stop()***
- ***HAL_DCMI_IRQHandler()***
- ***HAL_DCMI_ErrorCallback()***
- ***HAL_DCMI_LineEventCallback()***
- ***HAL_DCMI_VsyncEventCallback()***
- ***HAL_DCMI_FrameEventCallback()***

15.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.
- ***HAL_DCMI_ConfigCROP()***
- ***HAL_DCMI_DisableCROP()***
- ***HAL_DCMI_EnableCROP()***

15.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.
- ***HAL_DCMI_GetState()***
- ***HAL_DCMI_GetError()***

15.2.6 Initialization and Configuration functions

15.2.6.1 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**
Notes	• None.

15.2.6.2 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**
Notes	• None.

15.2.6.3 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.
Notes	• None.

15.2.6.4 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.
Notes	• None.

15.2.7 IO operation functions

15.2.7.1 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**
Notes	• None.

15.2.7.2 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**
Notes	• None.

15.2.7.3 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.
Notes	• None.

15.2.7.4 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.
Notes	• None.

15.2.7.5 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.

Notes	• None.

15.2.7.6 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.
Notes	• None.

15.2.7.7 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.
Notes	• None.

15.2.8 Peripheral Control functions

15.2.8.1 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. • **YSize :** DCMI Line number • **XSize :** DCMI Pixel per line • **X0 :** DCMI window X offset • **Y0 :** DCMI window Y offset
Return values	• **HAL status**
Notes	• None.

15.2.8.2 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**
Notes	• None.

15.2.8.3 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**
Notes	• None.

15.2.9 Peripheral State functions

15.2.9.1 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**
Notes	• None.

15.2.9.2 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**
Notes	• None.

15.3 DCMI Firmware driver defines

15.3.1 DCMI

DCMI

DCMI_Capture_Mode

- #define: ***DCMI_MODE_CONTINUOUS ((uint32_t)0x00000000)***

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

- #define: ***DCMI_MODE_SNAPSHOT ((uint32_t)DCMI_CR_CM)***

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

DCMI_Capture_Rate

- #define: ***DCMI_CR_ALL_FRAME ((uint32_t)0x00000000)***

All frames are captured

- #define: ***DCMI_CR_ALTERNATE_2_FRAME ((uint32_t)DCMI_CR_FCRC_0)***

Every alternate frame captured

- #define: ***DCMI_CR_ALTERNATE_4_FRAME ((uint32_t)DCMI_CR_FCRC_1)***

One frame in 4 frames captured

DCMI_Error_Code

- #define: ***HAL_DCMI_ERROR_NONE ((uint32_t)0x00000000)***

No error

- #define: ***HAL_DCMI_ERROR_OVF ((uint32_t)0x00000001)***

Overflow error

- #define: ***HAL_DCMI_ERROR_SYNC ((uint32_t)0x00000002)***

Synchronization error

- #define: ***HAL_DCMI_ERROR_TIMEOUT ((uint32_t)0x00000020)***

Timeout error

DCMI_Extended_Data_Mode

- #define: ***DCMI_EXTEND_DATA_8B ((uint32_t)0x00000000)***

Interface captures 8-bit data on every pixel clock

- #define: ***DCMI_EXTEND_DATA_10B ((uint32_t)DCMI_CR_EDM_0)***

Interface captures 10-bit data on every pixel clock

- #define: ***DCMI_EXTEND_DATA_12B ((uint32_t)DCMI_CR_EDM_1)***

Interface captures 12-bit data on every pixel clock

- #define: ***DCMI_EXTEND_DATA_14B ((uint32_t)(DCMI_CR_EDM_0 | DCMI_CR_EDM_1))***

Interface captures 14-bit data on every pixel clock

DCMI_Flags

- #define: ***DCMI_FLAG_HSYNC ((uint32_t)0x2001)***
- #define: ***DCMI_FLAG_VSYNC ((uint32_t)0x2002)***
- #define: ***DCMI_FLAG_FNE ((uint32_t)0x2004)***
- #define: ***DCMI_FLAG_FRAMERI ((uint32_t)DCMI_RISR_FRAME_RIS)***
- #define: ***DCMI_FLAG_OVFRI ((uint32_t)DCMI_RISR_OVF_RIS)***
- #define: ***DCMI_FLAG_ERRRI ((uint32_t)DCMI_RISR_ERR_RIS)***
- #define: ***DCMI_FLAG_VSYNCRI ((uint32_t)DCMI_RISR_VSYNC_RIS)***
- #define: ***DCMI_FLAG_LINERI ((uint32_t)DCMI_RISR_LINE_RIS)***
- #define: ***DCMI_FLAG_FRAMEMI ((uint32_t)0x1001)***
- #define: ***DCMI_FLAG_OVFMI ((uint32_t)0x1002)***
- #define: ***DCMI_FLAG_ERRMI ((uint32_t)0x1004)***

- #define: ***DCMI_FLAG_VSYNCMI ((uint32_t)0x1008)***

- #define: ***DCMI_FLAG_LINEMI ((uint32_t)0x1010)***

DCMI_HSYNC_Polarity

- #define: ***DCMI_HSPOLARITY_LOW ((uint32_t)0x00000000)***

Horizontal synchronization active Low

- #define: ***DCMI_HSPOLARITY_HIGH ((uint32_t)DCMI_CR_HSPOL)***

Horizontal synchronization active High

DCMI_interrupt_sources

- #define: ***DCMI_IT_FRAME ((uint32_t)DCMI_IER_FRAME_IE)***

- #define: ***DCMI_IT_OVF ((uint32_t)DCMI_IER_OVF_IE)***

- #define: ***DCMI_IT_ERR ((uint32_t)DCMI_IER_ERR_IE)***

- #define: ***DCMI_IT_VSYNC ((uint32_t)DCMI_IER_VSYNC_IE)***

- #define: ***DCMI_IT_LINE ((uint32_t)DCMI_IER_LINE_IE)***

DCMI_MODE_JPEG

- #define: ***DCMI_JPEG_DISABLE ((uint32_t)0x00000000)***

Mode JPEG Disabled

- #define: ***DCMI_JPEG_ENABLE ((uint32_t)DCMI_CR_JPEG)***

Mode JPEG Enabled

DCMI_PIXCK_Polarity

- #define: ***DCMI_PCKPOLARITY_FALLING ((uint32_t)0x00000000)***

Pixel clock active on Falling edge

- #define: ***DCMI_PCKPOLARITY_RISING ((uint32_t)DCMI_CR_PCKPOL)***

Pixel clock active on Rising edge

DCMI_Synchronization_Mode

- #define: ***DCMI_SYNCHRO_HARDWARE ((uint32_t)0x00000000)***

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

- #define: ***DCMI_SYNCHRO_EMBEDDED ((uint32_t)DCMI_CR_ESS)***

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

DCMI_VSYNC_Polarity

- #define: ***DCMI_VSPOLARITY_LOW ((uint32_t)0x00000000)***

Vertical synchronization active Low

- #define: ***DCMI_VSPOLARITY_HIGH ((uint32_t)DCMI_CR_VSPOL)***

Vertical synchronization active High

DCMI_Window_Coordinate

- #define: ***DCMI_WINDOW_COORDINATE ((uint32_t)0x3FFF)***

Window coordinate

DCMI_Window_Height

- #define: ***DCMI_WINDOW_HEIGHT ((uint32_t)0x1FFF)***

Window Height

16 HAL ETHERNET Generic Driver

16.1 ETH Firmware driver registers structures

16.1.1 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

16.1.2 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***

16.1.3 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 = 0xFFFFFFFF
- ***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 = 0xFFFFFFFF
- ***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 = 0xFFFF
- ***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

16.1.4 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

16.1.5 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***

16.1.6 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

16.1.7 ETH_TypeDef

ETH_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t MACCR***
- ***__IO uint32_t MACFFR***
- ***__IO uint32_t MACHTHR***
- ***__IO uint32_t MACHTLR***
- ***__IO uint32_t MACMIIAR***
- ***__IO uint32_t MACMIIDR***
- ***__IO uint32_t MACFCR***
- ***__IO uint32_t MACVLANTR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t MACRWUFFR***
- ***__IO uint32_t MACPMTCSR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t MACSR***
- ***__IO uint32_t MACIMR***
- ***__IO uint32_t MACA0HR***
- ***__IO uint32_t MACA0LR***
- ***__IO uint32_t MACA1HR***
- ***__IO uint32_t MACA1LR***
- ***__IO uint32_t MACA2HR***
- ***__IO uint32_t MACA2LR***
- ***__IO uint32_t MACA3HR***
- ***__IO uint32_t MACA3LR***
- ***uint32_t RESERVED2***
- ***__IO uint32_t MMCCR***
- ***__IO uint32_t MMCRIR***
- ***__IO uint32_t MMCTIR***
- ***__IO uint32_t MMCRIMR***
- ***__IO uint32_t MMCTIMR***
- ***uint32_t RESERVED3***
- ***__IO uint32_t MMCTGFSCCR***
- ***__IO uint32_t MMCTGFMSCCR***
- ***uint32_t RESERVED4***
- ***__IO uint32_t MMCTGFCR***
- ***uint32_t RESERVED5***
- ***__IO uint32_t MMCRFCECR***
- ***__IO uint32_t MMCRFAECR***
- ***uint32_t RESERVED6***
- ***__IO uint32_t MMCRGUFCR***
- ***uint32_t RESERVED7***
- ***__IO uint32_t PTPTSCR***

- ***__IO uint32_t PTPSSIR***
- ***__IO uint32_t PTPTSHR***
- ***__IO uint32_t PTPTSLR***
- ***__IO uint32_t PTPTSHUR***
- ***__IO uint32_t PTPTSLUR***
- ***__IO uint32_t PTPTSAR***
- ***__IO uint32_t PTPTTHR***
- ***__IO uint32_t PTPTTLR***
- ***__IO uint32_t RESERVED8***
- ***__IO uint32_t PTPTSSR***
- ***uint32_t RESERVED9***
- ***__IO uint32_t DMABMR***
- ***__IO uint32_t DMATPDR***
- ***__IO uint32_t DMARPDR***
- ***__IO uint32_t DMARDLAR***
- ***__IO uint32_t DMATDLAR***
- ***__IO uint32_t DMASR***
- ***__IO uint32_t DMAOMR***
- ***__IO uint32_t DMAIER***
- ***__IO uint32_t DMAMFBOCR***
- ***__IO uint32_t DMARSWTR***
- ***uint32_t RESERVED10***
- ***__IO uint32_t DMACHTDR***
- ***__IO uint32_t DMACHRDR***
- ***__IO uint32_t DMACHTBAR***
- ***__IO uint32_t DMACHRBAR***

Field Documentation

- ***__IO uint32_t ETH_TypeDef::MACCR***
- ***__IO uint32_t ETH_TypeDef::MACFFR***
- ***__IO uint32_t ETH_TypeDef::MACHTHR***
- ***__IO uint32_t ETH_TypeDef::MACHTLR***
- ***__IO uint32_t ETH_TypeDef::MACMIIAR***
- ***__IO uint32_t ETH_TypeDef::MACMIIDR***
- ***__IO uint32_t ETH_TypeDef::MACFCR***
- ***__IO uint32_t ETH_TypeDef::MACVLANTR***
- ***uint32_t ETH_TypeDef::RESERVED0[2]***
- ***__IO uint32_t ETH_TypeDef::MACRWUFFR***
- ***__IO uint32_t ETH_TypeDef::MACPMTCSR***
- ***uint32_t ETH_TypeDef::RESERVED1[2]***
- ***__IO uint32_t ETH_TypeDef::MACSR***
- ***__IO uint32_t ETH_TypeDef::MACIMR***
- ***__IO uint32_t ETH_TypeDef::MACA0HR***
- ***__IO uint32_t ETH_TypeDef::MACA0LR***
- ***__IO uint32_t ETH_TypeDef::MACA1HR***
- ***__IO uint32_t ETH_TypeDef::MACA1LR***
- ***__IO uint32_t ETH_TypeDef::MACA2HR***
- ***__IO uint32_t ETH_TypeDef::MACA2LR***
- ***__IO uint32_t ETH_TypeDef::MACA3HR***
- ***__IO uint32_t ETH_TypeDef::MACA3LR***

- ***uint32_t ETH_TypeDef::RESERVED2[40]***
- ***__IO uint32_t ETH_TypeDef::MMCCR***
- ***__IO uint32_t ETH_TypeDef::MMCRIR***
- ***__IO uint32_t ETH_TypeDef::MMCTIR***
- ***__IO uint32_t ETH_TypeDef::MMCRIMR***
- ***__IO uint32_t ETH_TypeDef::MMCTIMR***
- ***uint32_t ETH_TypeDef::RESERVED3[14]***
- ***__IO uint32_t ETH_TypeDef::MMCTGFSCCR***
- ***__IO uint32_t ETH_TypeDef::MMCTGFMSCCR***
- ***uint32_t ETH_TypeDef::RESERVED4[5]***
- ***__IO uint32_t ETH_TypeDef::MMCTGFCR***
- ***uint32_t ETH_TypeDef::RESERVED5[10]***
- ***__IO uint32_t ETH_TypeDef::MMCRFCECR***
- ***__IO uint32_t ETH_TypeDef::MMCRFAECR***
- ***uint32_t ETH_TypeDef::RESERVED6[10]***
- ***__IO uint32_t ETH_TypeDef::MMCRGUFCR***
- ***uint32_t ETH_TypeDef::RESERVED7[334]***
- ***__IO uint32_t ETH_TypeDef::PTPTSCR***
- ***__IO uint32_t ETH_TypeDef::PTPSSIR***
- ***__IO uint32_t ETH_TypeDef::PTPTSHR***
- ***__IO uint32_t ETH_TypeDef::PTPTSLR***
- ***__IO uint32_t ETH_TypeDef::PTPTSHUR***
- ***__IO uint32_t ETH_TypeDef::PTPTSLUR***
- ***__IO uint32_t ETH_TypeDef::PTPTSAR***
- ***__IO uint32_t ETH_TypeDef::PTPTTHR***
- ***__IO uint32_t ETH_TypeDef::PTPTTLR***
- ***__IO uint32_t ETH_TypeDef::RESERVED8***
- ***__IO uint32_t ETH_TypeDef::PTPTSSR***
- ***uint32_t ETH_TypeDef::RESERVED9[565]***
- ***__IO uint32_t ETH_TypeDef::DMABMR***
- ***__IO uint32_t ETH_TypeDef::DMATPDR***
- ***__IO uint32_t ETH_TypeDef::DMARPDR***
- ***__IO uint32_t ETH_TypeDef::DMARDLAR***
- ***__IO uint32_t ETH_TypeDef::DMATDLAR***
- ***__IO uint32_t ETH_TypeDef::DMASR***
- ***__IO uint32_t ETH_TypeDef::DMAOMR***
- ***__IO uint32_t ETH_TypeDef::DMAIER***
- ***__IO uint32_t ETH_TypeDef::DMAMFBOCR***
- ***__IO uint32_t ETH_TypeDef::DMARSWTR***
- ***uint32_t ETH_TypeDef::RESERVED10[8]***
- ***__IO uint32_t ETH_TypeDef::DMACHTDR***
- ***__IO uint32_t ETH_TypeDef::DMACHRDR***
- ***__IO uint32_t ETH_TypeDef::DMACHTBAR***
- ***__IO uint32_t ETH_TypeDef::DMACHRBAR***

16.2 ETH Firmware driver API description

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

16.2.1 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
      – __ETHMAC_CLK_ENABLE();
      – __ETHMACTX_CLK_ENABLE();
      – __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.

16.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the Ethernet peripheral
- De-initialize the Ethernet peripheral
- ***HAL_ETH_Init()***
- ***HAL_ETH_DeInit()***
- ***HAL_ETH_DMATxDescListInit()***
- ***HAL_ETH_DMARxDescListInit()***
- ***HAL_ETH_MspInit()***
- ***HAL_ETH_MspDeInit()***

16.2.3 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();
- ***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()***

16.2.4 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();
- ***HAL_ETH_Start()***
- ***HAL_ETH_Stop()***
- ***HAL_ETH_ConfigMAC()***
- ***HAL_ETH_ConfigDMA()***

16.2.5 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();
- ***HAL_ETH_GetState()***

16.2.6 Initialization and de-initialization functions

16.2.6.1 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**
Notes	• None.

16.2.6.2 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**
Notes	• None.

16.2.6.3 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**
Notes	• None.

16.2.6.4 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**
Notes	• None.

16.2.6.5 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.
Notes	• None.

16.2.6.6 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.
Notes	• None.

16.2.7 IO operation functions

16.2.7.1 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**
Notes	• None.

16.2.7.2 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**
Notes	• None.

16.2.7.3 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**
Notes	• None.

16.2.7.4 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**
Notes	• None.

16.2.7.5 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.
Notes	• None.

16.2.7.6 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.
Notes	• None.

16.2.7.7 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.
Notes	• None.

16.2.7.8 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**
Notes	• None.

16.2.7.9 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 More PHY register could be written depending on the used PHY • **RegValue :** the value to write
Return values	• **HAL status**
Notes	• None.

16.2.8 Peripheral Control functions

16.2.8.1 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**
Notes	• None.

16.2.8.2 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**
Notes	• None.

16.2.8.3 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**
Notes	• None.

16.2.8.4 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**
Notes	• None.

16.2.9 Peripheral State functions

16.2.9.1 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**
Notes	• None.

16.3 ETH Firmware driver defines

16.3.1 ETH

ETH

ETH_Address_Aligned_Beats

- #define: ***ETH_ADDRESSALIGNEDBEATS_ENABLE ((uint32_t)0x02000000)***
- #define: ***ETH_ADDRESSALIGNEDBEATS_DISABLE ((uint32_t)0x00000000)***

ETH_Automatic_Pad_CRC_Strip

- #define: ***ETH_AUTOMATICPADCRCSTRIP_ENABLE ((uint32_t)0x00000080)***
- #define: ***ETH_AUTOMATICPADCRCSTRIP_DISABLE ((uint32_t)0x00000000)***

ETH_AutoNegotiation

- #define: ***ETH_AUTONEGOTIATION_ENABLE ((uint32_t)0x00000001)***
- #define: ***ETH_AUTONEGOTIATION_DISABLE ((uint32_t)0x00000000)***

ETH_Back_Off_Limit

- #define: ***ETH_BACKOFFLIMIT_10 ((uint32_t)0x00000000)***
- #define: ***ETH_BACKOFFLIMIT_8 ((uint32_t)0x00000020)***

- #define: ***ETH_BACKOFFLIMIT_4 ((uint32_t)0x00000040)***

- #define: ***ETH_BACKOFFLIMIT_1 ((uint32_t)0x00000060)***

ETH_Broadcast_Frames_Reception

- #define: ***ETH_BROADCASTFRAMESRECEPTION_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_BROADCASTFRAMESRECEPTION_DISABLE ((uint32_t)0x00000020)***

ETH_Buffers_setting

- #define: ***ETH_MAX_PACKET_SIZE ((uint32_t)1524)***

ETH_HEADER + ETH_EXTRA + VLAN_TAG + MAX_ETH_PAYLOAD + ETH_CRC

- #define: ***ETH_HEADER ((uint32_t)14)***

6 byte Dest addr, 6 byte Src addr, 2 byte length/type

- #define: ***ETH_CRC ((uint32_t)4)***

Ethernet CRC

- #define: ***ETH_EXTRA ((uint32_t)2)***

Extra bytes in some cases

- #define: ***VLAN_TAG ((uint32_t)4)***

optional 802.1q VLAN Tag

- #define: ***MIN_ETH_PAYLOAD ((uint32_t)46)***

Minimum Ethernet payload size

- #define: ***MAX_ETH_PAYLOAD ((uint32_t)1500)***

Maximum Ethernet payload size

- #define: ***JUMBO_FRAME_PAYLOAD ((uint32_t)9000)***

Jumbo frame payload size

- #define: ***ETH_RX_BUF_SIZE ETH_MAX_PACKET_SIZE***

- #define: ***ETH_RXBUFNB ((uint32_t)5***

- #define: ***ETH_TX_BUF_SIZE ETH_MAX_PACKET_SIZE***

- #define: ***ETH_TXBUFNB ((uint32_t)5***

- #define: ***ETH_DMATXDESC_OWN ((uint32_t)0x80000000)***

OWN bit: descriptor is owned by DMA engine

- #define: ***ETH_DMATXDESC_IC ((uint32_t)0x40000000)***

Interrupt on Completion

- #define: ***ETH_DMATXDESC_LS ((uint32_t)0x20000000)***

Last Segment

- #define: ***ETH_DMATXDESC_FS ((uint32_t)0x10000000)***

First Segment

- #define: ***ETH_DMATXDESC_DC ((uint32_t)0x08000000)***

Disable CRC

- #define: ***ETH_DMATXDESC_DP ((uint32_t)0x04000000)***

Disable Padding

- #define: ***ETH_DMATXDESC_TTSE ((uint32_t)0x02000000)***

Transmit Time Stamp Enable

- #define: ***ETH_DMATXDESC_CIC ((uint32_t)0x00C00000)***

Checksum Insertion Control: 4 cases

- #define: ***ETH_DMATXDESC_CIC_BYPASS ((uint32_t)0x00000000)***

Do Nothing: Checksum Engine is bypassed

- #define: ***ETH_DMATXDESC_CIC_IPV4HEADER ((uint32_t)0x00400000)***

IPV4 header Checksum Insertion

- #define: ***ETH_DMATXDESC_CIC_TCPUDPICMP_SEGMENT ((uint32_t)0x00800000)***

TCP/UDP/ICMP Checksum Insertion calculated over segment only

- #define: ***ETH_DMATXDESC_CIC_TCPUDPICMP_FULL ((uint32_t)0x00C00000)***

TCP/UDP/ICMP Checksum Insertion fully calculated

- #define: ***ETH_DMATXDESC_TER ((uint32_t)0x00200000)***

Transmit End of Ring

- #define: ***ETH_DMATXDESC_TCH ((uint32_t)0x00100000)***

Second Address Chained

- #define: ***ETH_DMATXDESC_TTSS ((uint32_t)0x00020000)***

Tx Time Stamp Status

- #define: ***ETH_DMATXDESC_IHE ((uint32_t)0x00010000)***

IP Header Error

- #define: ***ETH_DMATXDESC_ES ((uint32_t)0x00008000)***

Error summary: OR of the following bits: UE || ED || EC || LCO || NC || LCA || FF || JT

- #define: ***ETH_DMATXDESC_JT ((uint32_t)0x00004000)***

Jabber Timeout

- #define: ***ETH_DMATXDESC_FF ((uint32_t)0x00002000)***

Frame Flushed: DMA/MTL flushed the frame due to SW flush

- #define: ***ETH_DMATXDESC_PCE ((uint32_t)0x00001000)***

Payload Checksum Error

- #define: ***ETH_DMATXDESC_LCA ((uint32_t)0x00000800)***

Loss of Carrier: carrier lost during transmission

- #define: ***ETH_DMATXDESC_NC ((uint32_t)0x00000400)***

No Carrier: no carrier signal from the transceiver

- #define: ***ETH_DMATXDESC_LCO ((uint32_t)0x00000200)***

Late Collision: transmission aborted due to collision

- #define: ***ETH_DMATXDESC_EC ((uint32_t)0x00000100)***

Excessive Collision: transmission aborted after 16 collisions

- #define: ***ETH_DMATXDESC_VF ((uint32_t)0x00000080)***

VLAN Frame

- #define: ***ETH_DMATXDESC_CC ((uint32_t)0x00000078)***

Collision Count

- #define: ***ETH_DMATXDESC_ED ((uint32_t)0x00000004)***

Excessive Deferral

- #define: ***ETH_DMATXDESC_UF ((uint32_t)0x00000002)***

Underflow Error: late data arrival from the memory

- #define: ***ETH_DMATXDESC_DB ((uint32_t)0x00000001)***

Deferred Bit

- #define: ***ETH_DMATXDESC_TBS2 ((uint32_t)0x1FFF0000)***

Transmit Buffer2 Size

- #define: ***ETH_DMATXDESC_TBS1 ((uint32_t)0x00001FFF)***

Transmit Buffer1 Size

- #define: ***ETH_DMATXDESC_B1AP ((uint32_t)0xFFFFFFFF)***

Buffer1 Address Pointer

- #define: ***ETH_DMATXDESC_B2AP ((uint32_t)0xFFFFFFFF)***

Buffer2 Address Pointer

- #define: ***ETH_DMAPTPTXDESC_TTSL ((uint32_t)0xFFFFFFFF)***

- #define: ***ETH_DMAPTPTXDESC_TTSH ((uint32_t)0xFFFFFFFF)***

ETH_Carrier_Sense

- #define: ***ETH_CARRIERSENCE_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_CARRIERSENCE_DISABLE ((uint32_t)0x00010000)***

ETH_Checksum_Mode

- #define: ***ETH_CHECKSUM_BY_HARDWARE ((uint32_t)0x00000000)***

- #define: ***ETH_CHECKSUM_BY_SOFTWARE ((uint32_t)0x00000001)***

ETH_Checksum_Offload

- #define: ***ETH_CHECKSUMOFFLAOD_ENABLE ((uint32_t)0x00000400)***

- #define: ***ETH_CHECKSUMOFFLAOD_DISABLE ((uint32_t)0x00000000)***

ETH_Deferral_Check

- #define: ***ETH_DEFFERRALCHECK_ENABLE ((uint32_t)0x00000010)***
- #define: ***ETH_DEFFERRALCHECK_DISABLE ((uint32_t)0x00000000)***

ETH_Destination_Addr_Filter

- #define: ***ETH_DESTINATIONADDRFILTER_NORMAL ((uint32_t)0x00000000)***
- #define: ***ETH_DESTINATIONADDRFILTER_INVERSE ((uint32_t)0x00000008)***

ETH_DMA_Arbitration

- #define: ***ETH_DMAARBITRATION_ROUNDROBIN_RXTX_1_1 ((uint32_t)0x00000000)***
- #define: ***ETH_DMAARBITRATION_ROUNDROBIN_RXTX_2_1 ((uint32_t)0x00004000)***
- #define: ***ETH_DMAARBITRATION_ROUNDROBIN_RXTX_3_1 ((uint32_t)0x00008000)***
- #define: ***ETH_DMAARBITRATION_ROUNDROBIN_RXTX_4_1 ((uint32_t)0x0000C000)***
- #define: ***ETH_DMAARBITRATION_RXPRIORTX ((uint32_t)0x00000002)***

ETH_DMA_Enhanced_descriptor_format

- #define: ***ETH_DMAENHANCEDDESCRIPTOR_ENABLE ((uint32_t)0x00000080)***
- #define: ***ETH_DMAENHANCEDDESCRIPTOR_DISABLE ((uint32_t)0x00000000)***

ETH_DMA_Flags

- #define: ***ETH_DMA_FLAG_TST ((uint32_t)0x20000000)***

Time-stamp trigger interrupt (on DMA)

- #define: ***ETH_DMA_FLAG_PMT ((uint32_t)0x10000000)***

PMT interrupt (on DMA)

- #define: ***ETH_DMA_FLAG_MMC ((uint32_t)0x08000000)***

MMC interrupt (on DMA)

- #define: ***ETH_DMA_FLAG_DATATRANSFERERROR ((uint32_t)0x00800000)***

Error bits 0-Rx DMA, 1-Tx DMA

- #define: ***ETH_DMA_FLAG_READWRITEERROR ((uint32_t)0x01000000)***

Error bits 0-write trnsf, 1-read transfr

- #define: ***ETH_DMA_FLAG_ACCESSERROR ((uint32_t)0x02000000)***

Error bits 0-data buffer, 1-desc. access

- #define: ***ETH_DMA_FLAG_NIS ((uint32_t)0x00010000)***

Normal interrupt summary flag

- #define: ***ETH_DMA_FLAG_AIS ((uint32_t)0x00008000)***

Abnormal interrupt summary flag

- #define: ***ETH_DMA_FLAG_ER ((uint32_t)0x00004000)***

Early receive flag

- #define: ***ETH_DMA_FLAG_FBE ((uint32_t)0x00002000)***

Fatal bus error flag

- #define: ***ETH_DMA_FLAG_ET ((uint32_t)0x00000400)***

Early transmit flag

- #define: ***ETH_DMA_FLAG_RWT ((uint32_t)0x00000200)***

Receive watchdog timeout flag

- #define: ***ETH_DMA_FLAG_RPS ((uint32_t)0x00000100)***

Receive process stopped flag

- #define: ***ETH_DMA_FLAG_RBU ((uint32_t)0x00000080)***

Receive buffer unavailable flag

- #define: ***ETH_DMA_FLAG_R ((uint32_t)0x00000040)***

Receive flag

- #define: ***ETH_DMA_FLAG_TU ((uint32_t)0x00000020)***

Underflow flag

- #define: ***ETH_DMA_FLAG_RO ((uint32_t)0x00000010)***

Overflow flag

- #define: ***ETH_DMA_FLAG_TJT ((uint32_t)0x00000008)***

Transmit jabber timeout flag

- #define: ***ETH_DMA_FLAG_TBU ((uint32_t)0x00000004)***

Transmit buffer unavailable flag

- #define: ***ETH_DMA_FLAG_TPS ((uint32_t)0x00000002)***

Transmit process stopped flag

- #define: ***ETH_DMA_FLAG_T ((uint32_t)0x00000001)***

Transmit flag

ETH_DMA_Interrupts

- #define: ***ETH_DMA_IT_TST ((uint32_t)0x20000000)***

Time-stamp trigger interrupt (on DMA)

- #define: ***ETH_DMA_IT_PMT ((uint32_t)0x10000000)***

PMT interrupt (on DMA)

- #define: ***ETH_DMA_IT_MMC ((uint32_t)0x08000000)***

MMC interrupt (on DMA)

- #define: ***ETH_DMA_IT_NIS ((uint32_t)0x00010000)***

Normal interrupt summary

- #define: ***ETH_DMA_IT_AIS ((uint32_t)0x00008000)***

Abnormal interrupt summary

- #define: ***ETH_DMA_IT_ER ((uint32_t)0x00004000)***

Early receive interrupt

- #define: ***ETH_DMA_IT_FBE ((uint32_t)0x00002000)***

Fatal bus error interrupt

- #define: ***ETH_DMA_IT_ET ((uint32_t)0x00000400)***

Early transmit interrupt

- #define: ***ETH_DMA_IT_RWT ((uint32_t)0x00000200)***

Receive watchdog timeout interrupt

- #define: ***ETH_DMA_IT_RPS ((uint32_t)0x00000100)***

Receive process stopped interrupt

- #define: ***ETH_DMA_IT_RBU ((uint32_t)0x00000080)***

Receive buffer unavailable interrupt

- #define: ***ETH_DMA_IT_R ((uint32_t)0x00000040)***

Receive interrupt

- #define: ***ETH_DMA_IT_TU ((uint32_t)0x00000020)***

Underflow interrupt

- #define: ***ETH_DMA_IT_RO ((uint32_t)0x00000010)***

Overflow interrupt

- #define: ***ETH_DMA_IT_TJT ((uint32_t)0x00000008)***

Transmit jabber timeout interrupt

- #define: ***ETH_DMA_IT_TBU ((uint32_t)0x00000004)***

Transmit buffer unavailable interrupt

- #define: ***ETH_DMA_IT_TPS ((uint32_t)0x00000002)***

Transmit process stopped interrupt

- #define: ***ETH_DMA_IT_T ((uint32_t)0x00000001)***

Transmit interrupt

ETH_DMA_overflow_

- #define: ***ETH_DMA_OVERFLOW_RXFIFOCOUNTER ((uint32_t)0x10000000)***

Overflow bit for FIFO overflow counter

- #define: ***ETH_DMA_OVERFLOW_MISSEDFRAMECOUNTER ((uint32_t)0x00010000)***

Overflow bit for missed frame counter

ETH_DMA_receive_process_state_

- #define: ***ETH_DMA_RECEIVEPROCESS_STOPPED ((uint32_t)0x00000000)***

Stopped - Reset or Stop Rx Command issued

- #define: ***ETH_DMA_RECEIVEPROCESS_FETCHING ((uint32_t)0x00020000)***

Running - fetching the Rx descriptor

- #define: ***ETH_DMA_RECEIVEPROCESS_WAITING ((uint32_t)0x00060000)***

Running - waiting for packet

- #define: ***ETH_DMA_RECEIVEPROCESS_SUSPENDED ((uint32_t)0x00080000)***

Suspended - Rx Descriptor unavailable

- #define: ***ETH_DMA_RECEIVEPROCESS_CLOSING ((uint32_t)0x000A0000)***

Running - closing descriptor

- #define: ***ETH_DMA_RECEIVEPROCESS_QUEUING ((uint32_t)0x000E0000)***

Running - queuing the receive frame into host memory

ETH_DMA_Rx_descriptor

- #define: ***ETH_DMARXDESC_OWN ((uint32_t)0x80000000)***

OWN bit: descriptor is owned by DMA engine

- #define: ***ETH_DMARXDESC_AFM ((uint32_t)0x40000000)***

DA Filter Fail for the rx frame

- #define: ***ETH_DMARXDESC_FL ((uint32_t)0x3FFF0000)***

Receive descriptor frame length

- #define: ***ETH_DMARXDESC_ES ((uint32_t)0x00008000)***

Error summary: OR of the following bits: DE || OE || IPC || LC || RWT || RE || CE

- #define: ***ETH_DMARXDESC_DE ((uint32_t)0x00004000)***

Descriptor error: no more descriptors for receive frame

- #define: ***ETH_DMARXDESC_SAF ((uint32_t)0x00002000)***

SA Filter Fail for the received frame

- #define: ***ETH_DMARXDESC_LE ((uint32_t)0x00001000)***

Frame size not matching with length field

- #define: ***ETH_DMARXDESC_OE ((uint32_t)0x00000800)***

Overflow Error: Frame was damaged due to buffer overflow

- #define: ***ETH_DMARXDESC_VLAN ((uint32_t)0x00000400)***

VLAN Tag: received frame is a VLAN frame

- #define: ***ETH_DMARXDESC_FS ((uint32_t)0x00000200)***

First descriptor of the frame

- #define: ***ETH_DMARXDESC_LS ((uint32_t)0x00000100)***

Last descriptor of the frame

- #define: ***ETH_DMARXDESC_IPV4HCE ((uint32_t)0x00000080)***

IPC Checksum Error: Rx Ipv4 header checksum error

- #define: ***ETH_DMARXDESC_LC ((uint32_t)0x00000040)***

Late collision occurred during reception

- #define: ***ETH_DMARXDESC_FT ((uint32_t)0x00000020)***

Frame type - Ethernet, otherwise 802.3

- #define: ***ETH_DMARXDESC_RWT ((uint32_t)0x00000010)***

Receive Watchdog Timeout: watchdog timer expired during reception

- #define: ***ETH_DMARXDESC_RE ((uint32_t)0x00000008)***

Receive error: error reported by MII interface

- #define: ***ETH_DMARXDESC_DBE ((uint32_t)0x00000004)***

Dribble bit error: frame contains non int multiple of 8 bits

- #define: ***ETH_DMARXDESC_CE ((uint32_t)0x00000002)***

CRC error

- #define: ***ETH_DMARXDESC_MAMPCE ((uint32_t)0x00000001)***

Rx MAC Address/Payload Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error

- #define: ***ETH_DMARXDESC_DIC ((uint32_t)0x80000000)***

Disable Interrupt on Completion

- #define: ***ETH_DMARXDESC_RBS2 ((uint32_t)0x1FFF0000)***

Receive Buffer2 Size

- #define: ***ETH_DMARXDESC_RER ((uint32_t)0x00008000)***

Receive End of Ring

- #define: ***ETH_DMARXDESC_RCH ((uint32_t)0x00004000)***

Second Address Chained

- #define: ***ETH_DMARXDESC_RBS1 ((uint32_t)0x00001FFF)***

Receive Buffer1 Size

- #define: ***ETH_DMARXDESC_B1AP ((uint32_t)0xFFFFFFFF)***

Buffer1 Address Pointer

- #define: ***ETH_DMARXDESC_B2AP ((uint32_t)0xFFFFFFFF)***

Buffer2 Address Pointer

- #define: ***ETH_DMAPTPRXDESC_PTPV ((uint32_t)0x00002000)***

- #define: ***ETH_DMAPTPRXDESC_PTPFT ((uint32_t)0x00001000)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT ((uint32_t)0x00000F00)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_SYNC ((uint32_t)0x00000100)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_FOLLOWUP ((uint32_t)0x00000200)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_DELAYREQ ((uint32_t)0x00000300)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_DELAYRESP ((uint32_t)0x00000400)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_PDELAYREQ_ANNOUNCE ((uint32_t)0x00000500)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_PDELAYRESP_MANAG ((uint32_t)0x00000600)***

- #define: ***ETH_DMAPTPRXDESC_PTPMT_PDELAYRESPFOLLOWUP_SIGNAL ((uint32_t)0x00000700)***
- #define: ***ETH_DMAPTPRXDESC_IPV6PR ((uint32_t)0x00000080)***
- #define: ***ETH_DMAPTPRXDESC_IPV4PR ((uint32_t)0x00000040)***
- #define: ***ETH_DMAPTPRXDESC_IPCB ((uint32_t)0x00000020)***
- #define: ***ETH_DMAPTPRXDESC_IPPE ((uint32_t)0x00000010)***
- #define: ***ETH_DMAPTPRXDESC_IPHE ((uint32_t)0x00000008)***
- #define: ***ETH_DMAPTPRXDESC_IPPT ((uint32_t)0x00000007)***
- #define: ***ETH_DMAPTPRXDESC_IPPT_UDP ((uint32_t)0x00000001)***
- #define: ***ETH_DMAPTPRXDESC_IPPT_TCP ((uint32_t)0x00000002)***
- #define: ***ETH_DMAPTPRXDESC_IPPT_ICMP ((uint32_t)0x00000003)***
- #define: ***ETH_DMAPTPRXDESC_RTSL ((uint32_t)0xFFFFFFFF)***
- #define: ***ETH_DMAPTPRXDESC_RTSH ((uint32_t)0xFFFFFFFF)***

ETH_DMA_Rx_descriptor_buffers_

- #define: ***ETH_DMARXDESC_BUFFER1 ((uint32_t)0x00000000)***

DMA Rx Desc Buffer1

- #define: ***ETH_DMARXDESC_BUFFER2 ((uint32_t)0x00000001)***

DMA Rx Desc Buffer2

- #define: ***ETH_DMATXDESC_COLLISION_COUNTSHIFT ((uint32_t)3)***

- #define: ***ETH_DMATXDESC_BUFFER2_SIZESHIFT ((uint32_t)16)***

- #define: ***ETH_DMARXDESC_FRAME_LENGTHSHIFT ((uint32_t)16)***

- #define: ***ETH_DMARXDESC_BUFFER2_SIZESHIFT ((uint32_t)16)***

- #define: ***ETH_DMARXDESC_FRAMELENGTHSHIFT ((uint32_t)16)***

ETH_DMA_transmit_process_state_

- #define: ***ETH_DMA_TRANSMITPROCESS_STOPPED ((uint32_t)0x00000000)***

Stopped - Reset or Stop Tx Command issued

- #define: ***ETH_DMA_TRANSMITPROCESS_FETCHING ((uint32_t)0x00100000)***

Running - fetching the Tx descriptor

- #define: ***ETH_DMA_TRANSMITPROCESS_WAITING ((uint32_t)0x00200000)***

Running - waiting for status

- #define: ***ETH_DMA_TRANSMITPROCESS_READING ((uint32_t)0x00300000)***

Running - reading the data from host memory

- #define: ***ETH_DMA_TRANSMITPROCESS_SUSPENDED ((uint32_t)0x00600000)***

Suspended - Tx Descriptor unavailable

- #define: ***ETH_DMA_TRANSMITPROCESS_CLOSING ((uint32_t)0x00700000)***

Running - closing Rx descriptor

ETH_DMA_Tx_descriptor_Checksum_Insertion_Control

- #define: ***ETH_DMATXDESC_CHECKSUMBYPASS ((uint32_t)0x00000000)***

Checksum engine bypass

- #define: ***ETH_DMATXDESC_CHECKSUMIPV4HEADER ((uint32_t)0x00400000)***

IPv4 header checksum insertion

- #define: ***ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT ((uint32_t)0x00800000)***

TCP/UDP/ICMP checksum insertion. Pseudo header checksum is assumed to be present

- #define: ***ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL ((uint32_t)0x00C00000)***

TCP/UDP/ICMP checksum fully in hardware including pseudo header

ETH_DMA_Tx_descriptor_segment

- #define: ***ETH_DMATXDESC_LASTSEGMENTS ((uint32_t)0x40000000)***

Last Segment

- #define: ***ETH_DMATXDESC_FIRSTSEGMENT ((uint32_t)0x20000000)***

First Segment

ETH_Drop_TCP_IP_Checksum_Error_Frame

- #define: ***ETH_DROPTCPIPCHECKSUMERRORFRAME_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_DROPTCPIPCHECKSUMERRORFRAME_DISABLE ((uint32_t)0x04000000)***

ETH_Duplex_Mode

- #define: ***ETH_MODE_FULLDUPLEX ((uint32_t)0x00000800)***

- #define: ***ETH_MODE_HALFDUPLEX ((uint32_t)0x00000000)***

ETH_Fixed_Burst

- #define: ***ETH_FIXEDBURST_ENABLE ((uint32_t)0x00010000)***

- #define: ***ETH_FIXEDBURST_DISABLE ((uint32_t)0x00000000)***

ETH_Flush_Received_Frame

- #define: ***ETH_FLUSHRECEIVEDFRAME_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_FLUSHRECEIVEDFRAME_DISABLE ((uint32_t)0x01000000)***

ETH_Forward_Error_Frames

- #define: ***ETH_FORWARDERRORFRAMES_ENABLE ((uint32_t)0x00000080)***

- #define: ***ETH_FORWARDERRORFRAMES_DISABLE ((uint32_t)0x00000000)***

ETH_Forward_Undersized_Good_Frames

- #define: ***ETH_FORWARDUNDERSIZEDGOODFRAMES_ENABLE ((uint32_t)0x00000040)***

- #define: ***ETH_FORWARDUNDERSIZEDGOODFRAMES_DISABLE ((uint32_t)0x00000000)***

ETH_Inter_Frame_Gap

- #define: ***ETH_INTERFRAMEGAP_96BIT ((uint32_t)0x00000000)***

minimum IFG between frames during transmission is 96Bit

- #define: ***ETH_INTERFRAMEGAP_88BIT ((uint32_t)0x00020000)***

minimum IFG between frames during transmission is 88Bit

- #define: ***ETH_INTERFRAMEGAP_80BIT ((uint32_t)0x00040000)***

minimum IFG between frames during transmission is 80Bit

- #define: ***ETH_INTERFRAMEGAP_72BIT ((uint32_t)0x00060000)***

minimum IFG between frames during transmission is 72Bit

- #define: ***ETH_INTERFRAMEGAP_64BIT ((uint32_t)0x00080000)***

minimum IFG between frames during transmission is 64Bit

- #define: ***ETH_INTERFRAMEGAP_56BIT ((uint32_t)0x000A0000)***

minimum IFG between frames during transmission is 56Bit

- #define: ***ETH_INTERFRAMEGAP_48BIT ((uint32_t)0x000C0000)***

minimum IFG between frames during transmission is 48Bit

- #define: ***ETH_INTERFRAMEGAP_40BIT ((uint32_t)0x000E0000)***

minimum IFG between frames during transmission is 40Bit

ETH_Jabber

- #define: ***ETH_JABBER_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_JABBER_DISABLE ((uint32_t)0x00400000)***

ETH_Loop_Back_Mode

- #define: ***ETH_LOOPBACKMODE_ENABLE ((uint32_t)0x00001000)***

- #define: ***ETH_LOOPBACKMODE_DISABLE ((uint32_t)0x00000000)***

ETH_MAC_addresses

- #define: ***ETH_MAC_ADDRESS0 ((uint32_t)0x00000000)***

- #define: ***ETH_MAC_ADDRESS1 ((uint32_t)0x00000008)***

- #define: ***ETH_MAC_ADDRESS2 ((uint32_t)0x00000010)***

- #define: ***ETH_MAC_ADDRESS3 ((uint32_t)0x00000018)***

ETH_MAC_addresses_filter_Mask_bytes

- #define: ***ETH_MAC_ADDRESSMASK_BYTE6 ((uint32_t)0x20000000)***

Mask MAC Address high reg bits [15:8]

- #define: ***ETH_MAC_ADDRESSMASK_BYTE5 ((uint32_t)0x10000000)***

Mask MAC Address high reg bits [7:0]

- #define: ***ETH_MAC_ADDRESSMASK_BYTE4 ((uint32_t)0x08000000)***

Mask MAC Address low reg bits [31:24]

- #define: ***ETH_MAC_ADDRESSMASK_BYTE3 ((uint32_t)0x04000000)***

Mask MAC Address low reg bits [23:16]

- #define: ***ETH_MAC_ADDRESSMASK_BYTE2 ((uint32_t)0x02000000)***

Mask MAC Address low reg bits [15:8]

- #define: ***ETH_MAC_ADDRESSMASK_BYTE1 ((uint32_t)0x01000000)***

Mask MAC Address low reg bits [70]

ETH_MAC_addresses_filter_SA_DA_filed_of_received_frames

- #define: ***ETH_MAC_ADDRESSFILTER_SA ((uint32_t)0x00000000)***
- #define: ***ETH_MAC_ADDRESSFILTER_DA ((uint32_t)0x00000008)***

ETH_MAC_Debug_flags

- #define: ***ETH_MAC_TXFIFO_FULL ((uint32_t)0x02000000)***
- #define: ***ETH_MAC_TXFIFONOT_EMPTY ((uint32_t)0x01000000)***
- #define: ***ETH_MAC_TXFIFO_WRITE_ACTIVE ((uint32_t)0x00400000)***
- #define: ***ETH_MAC_TXFIFO_IDLE ((uint32_t)0x00000000)***
- #define: ***ETH_MAC_TXFIFO_READ ((uint32_t)0x00100000)***
- #define: ***ETH_MAC_TXFIFO_WAITING ((uint32_t)0x00200000)***
- #define: ***ETH_MAC_TXFIFO_WRITING ((uint32_t)0x00300000)***
- #define: ***ETH_MAC_TRANSMISSION_PAUSE ((uint32_t)0x00080000)***
- #define: ***ETH_MAC_TRANSMITFRAMECONTROLLER_IDLE ((uint32_t)0x00000000)***
- #define: ***ETH_MAC_TRANSMITFRAMECONTROLLER_WAITING ((uint32_t)0x00020000)***

- #define: ***ETH_MAC_TRANSMITFRAMECONTROLLER_GENRATING_PCF ((uint32_t)0x00040000)***
- #define: ***ETH_MAC_TRANSMITFRAMECONTROLLER_TRANSFERRING ((uint32_t)0x00060000)***
- #define: ***ETH_MAC_MII_TRANSMIT_ACTIVE ((uint32_t)0x00010000)***
- #define: ***ETH_MAC_RXFIFO_EMPTY ((uint32_t)0x00000000)***
- #define: ***ETH_MAC_RXFIFO_BELOW_THRESHOLD ((uint32_t)0x00000100)***
- #define: ***ETH_MAC_RXFIFO_ABOVE_THRESHOLD ((uint32_t)0x00000200)***
- #define: ***ETH_MAC_RXFIFO_FULL ((uint32_t)0x00000300)***
- #define: ***ETH_MAC_READCONTROLLER_IDLE ((uint32_t)0x00000060)***
- #define: ***ETH_MAC_READCONTROLLER_READING_DATA ((uint32_t)0x00000060)***
- #define: ***ETH_MAC_READCONTROLLER_READING_STATUS ((uint32_t)0x00000060)***
- #define: ***ETH_MAC_READCONTROLLER_ FLUSHING ((uint32_t)0x00000060)***

- #define: ***ETH_MAC_RXFIFO_WRITE_ACTIVE ((uint32_t)0x00000010)***

- #define: ***ETH_MAC_SMALL_FIFO_NOTACTIVE ((uint32_t)0x00000000)***

- #define: ***ETH_MAC_SMALL_FIFO_READ_ACTIVE ((uint32_t)0x00000002)***

- #define: ***ETH_MAC_SMALL_FIFO_WRITE_ACTIVE ((uint32_t)0x00000004)***

- #define: ***ETH_MAC_SMALL_FIFO_RW_ACTIVE ((uint32_t)0x00000006)***

- #define: ***ETH_MAC_MII_RECEIVE_PROTOCOL_AVTIVE ((uint32_t)0x00000001)***

ETH_MAC_Flags

- #define: ***ETH_MAC_FLAG_TST ((uint32_t)0x00000200)***

Time stamp trigger flag (on MAC)

- #define: ***ETH_MAC_FLAG_MMCT ((uint32_t)0x00000040)***

MMC transmit flag

- #define: ***ETH_MAC_FLAG_MMCR ((uint32_t)0x00000020)***

MMC receive flag

- #define: ***ETH_MAC_FLAG_MMC ((uint32_t)0x00000010)***

MMC flag (on MAC)

- #define: ***ETH_MAC_FLAG_PMT ((uint32_t)0x00000008)***

PMT flag (on MAC)

ETH_MAC_Interrupts

- #define: ***ETH_MAC_IT_TST ((uint32_t)0x00000200)***

Time stamp trigger interrupt (on MAC)

- #define: ***ETH_MAC_IT_MMCT ((uint32_t)0x00000040)***

MMC transmit interrupt

- #define: ***ETH_MAC_IT_MMCR ((uint32_t)0x00000020)***

MMC receive interrupt

- #define: ***ETH_MAC_IT_MMC ((uint32_t)0x00000010)***

MMC interrupt (on MAC)

- #define: ***ETH_MAC_IT_PMT ((uint32_t)0x00000008)***

PMT interrupt (on MAC)

ETH_Media_Interface

- #define: ***ETH_MEDIA_INTERFACE_MII ((uint32_t)0x00000000)***

- #define: ***ETH_MEDIA_INTERFACE_RMII ((uint32_t)SYSCFG_PMC_MII_RMII_SEL)***

ETH_MMC_Registers

- #define: ***ETH_MMCCR ((uint32_t)0x00000100)***

MMC CR register

- #define: ***ETH_MMCRIR ((uint32_t)0x00000104)***

MMC RIR register

- #define: ***ETH_MMCTIR ((uint32_t)0x00000108)***

MMC TIR register

- #define: ***ETH_MMCRIMR ((uint32_t)0x0000010C)***

MMC RIMR register

- #define: ***ETH_MMCTIMR ((uint32_t)0x00000110)***

MMC TIMR register

- #define: ***ETH_MMCTGFSCCR ((uint32_t)0x0000014C)***

MMC TGFSCCR register

- #define: ***ETH_MMCTGFMSCCR ((uint32_t)0x00000150)***

MMC TGFMSCCR register

- #define: ***ETH_MMCTGFCR ((uint32_t)0x00000168)***

MMC TGFCR register

- #define: ***ETH_MMCRFCECR ((uint32_t)0x00000194)***

MMC RFCECR register

- #define: ***ETH_MMCRFAECR ((uint32_t)0x00000198)***

MMC RFAECR register

- #define: ***ETH_MMCRGUFCR ((uint32_t)0x000001C4)***

MMC RGUFCR register

ETH_MMC_Rx_Interrupts

- #define: ***ETH_MMC_IT_RGUF ((uint32_t)0x10020000)***

When Rx good unicast frames counter reaches half the maximum value

- #define: ***ETH_MMC_IT_RFAE ((uint32_t)0x10000040)***

When Rx alignment error counter reaches half the maximum value

- #define: ***ETH_MMC_IT_RFCE ((uint32_t)0x10000020)***

When Rx crc error counter reaches half the maximum value

ETH_MMC_Tx_Interrupts

- #define: ***ETH_MMC_IT_TGF ((uint32_t)0x00200000)***

When Tx good frame counter reaches half the maximum value

- #define: ***ETH_MMC_IT_TGFMSC ((uint32_t)0x00008000)***

When Tx good multi col counter reaches half the maximum value

- #define: ***ETH_MMC_IT_TGFSC ((uint32_t)0x00004000)***

When Tx good single col counter reaches half the maximum value

ETH_Multicast_Frames_Filter

- #define: ***ETH_MULTICASTFRAMESFILTER_PERFECTHASHTABLE ((uint32_t)0x00000404)***

- #define: ***ETH_MULTICASTFRAMESFILTER_HASHTABLE ((uint32_t)0x00000004)***

- #define: ***ETH_MULTICASTFRAMESFILTER_PERFECT ((uint32_t)0x00000000)***

- #define: ***ETH_MULTICASTFRAMESFILTER_NONE ((uint32_t)0x00000010)***

ETH_Pass_Control_Frames

- #define: ***ETH_PASSCONTROLFRAMES_BLOCKALL ((uint32_t)0x00000040)***

MAC filters all control frames from reaching the application

- #define: ***ETH_PASSCONTROLFRAMES_FORWARDALL ((uint32_t)0x00000080)***

MAC forwards all control frames to application even if they fail the Address Filter

- #define: ***ETH_PASSCONTROLFRAMES_FORWARDPASSEDADDRFILTER ((uint32_t)0x000000C0)***

MAC forwards control frames that pass the Address Filter.

ETH_Pause_Low_Threshold

- #define: ***ETH_PAUSELOWTHRESHOLD_MINUS4 ((uint32_t)0x00000000)***

Pause time minus 4 slot times

- #define: ***ETH_PAUSELOWTHRESHOLD_MINUS28 ((uint32_t)0x00000010)***

Pause time minus 28 slot times

- #define: ***ETH_PAUSELOWTHRESHOLD_MINUS144 ((uint32_t)0x00000020)***

Pause time minus 144 slot times

- #define: ***ETH_PAUSELOWTHRESHOLD_MINUS256 ((uint32_t)0x00000030)***

Pause time minus 256 slot times

ETH_PMT_Flags

- #define: ***ETH_PMT_FLAG_WUFFRPR ((uint32_t)0x80000000)***

Wake-Up Frame Filter Register Pointer Reset

- #define: ***ETH_PMT_FLAG_WUFR ((uint32_t)0x00000040)***

Wake-Up Frame Received

- #define: ***ETH_PMT_FLAG_MPR ((uint32_t)0x00000020)***

Magic Packet Received

ETH_Promiscuous_Mode

- #define: ***ETH_PROMISCIOUSMODE_ENABLE ((uint32_t)0x00000001)***

- #define: ***ETH_PROMISCIOUSMODE_DISABLE ((uint32_t)0x00000000)***

ETH_Receive_All

- #define: ***ETH_RECEIVEALL_ENABLE ((uint32_t)0x80000000)***

- #define: ***ETH_RECEIVEAll_DISABLE ((uint32_t)0x00000000)***

ETH_Receive_Flow_Control

- #define: ***ETH_RECEIVEFLOWCONTROL_ENABLE ((uint32_t)0x00000004)***

- #define: ***ETH_RECEIVEFLOWCONTROL_DISABLE ((uint32_t)0x00000000)***

ETH_Receive_Own

- #define: ***ETH_RECEIVEOWN_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_RECEIVEOWN_DISABLE ((uint32_t)0x00002000)***

ETH_Receive_Store_Forward

- #define: ***ETH_RECEIVESTOREFORWARD_ENABLE ((uint32_t)0x02000000)***

- #define: ***ETH_RECEIVESTOREFORWARD_DISABLE ((uint32_t)0x00000000)***

ETH_Receive_Threshold_Control

- #define: ***ETH_RECEIVEDTHRESHOLDCONTROL_64BYTES ((uint32_t)0x00000000)***

threshold level of the MTL Receive FIFO is 64 Bytes

- #define: ***ETH_RECEIVEDTHRESHOLDCONTROL_32BYTES ((uint32_t)0x00000008)***

threshold level of the MTL Receive FIFO is 32 Bytes

- #define: ***ETH_RECEIVEDTHRESHOLDCONTROL_96BYTES ((uint32_t)0x00000010)***

threshold level of the MTL Receive FIFO is 96 Bytes

- #define: ***ETH_RECEIVEDTHRESHOLDCONTROL_128BYTES ((uint32_t)0x00000018)***

threshold level of the MTL Receive FIFO is 128 Bytes

ETH_Retry_Transmission

- #define: ***ETH_RETRYTRANSMISSION_ENABLE ((uint32_t)0x00000000)***

- #define: ***ETH_RETRYTRANSMISSION_DISABLE ((uint32_t)0x00000200)***

ETH_Rx_DMA_Burst_Length

- #define: ***ETH_RXDMABURSTLENGTH_1BEAT ((uint32_t)0x00020000)***

maximum number of beats to be transferred in one RxDMA transaction is 1

- #define: ***ETH_RXDMABURSTLENGTH_2BEAT ((uint32_t)0x00040000)***

maximum number of beats to be transferred in one RxDMA transaction is 2

- #define: ***ETH_RXDMABURSTLENGTH_4BEAT ((uint32_t)0x00080000)***

maximum number of beats to be transferred in one RxDMA transaction is 4

- #define: ***ETH_RXDMABURSTLENGTH_8BEAT ((uint32_t)0x00100000)***

maximum number of beats to be transferred in one RxDMA transaction is 8

- #define: ***ETH_RXDMABURSTLENGTH_16BEAT ((uint32_t)0x00200000)***

maximum number of beats to be transferred in one RxDMA transaction is 16

- #define: ***ETH_RXDMABURSTLENGTH_32BEAT ((uint32_t)0x00400000)***

maximum number of beats to be transferred in one RxDMA transaction is 32

- #define: ***ETH_RXDMABURSTLENGTH_4XPBL_4BEAT ((uint32_t)0x01020000)***

maximum number of beats to be transferred in one RxDMA transaction is 4

- #define: ***ETH_RXDMABURSTLENGTH_4XPBL_8BEAT ((uint32_t)0x01040000)***

maximum number of beats to be transferred in one RxDMA transaction is 8

- #define: ***ETH_RXDMABURSTLENGTH_4XPBL_16BEAT ((uint32_t)0x01080000)***

maximum number of beats to be transferred in one RxDMA transaction is 16

- #define: ***ETH_RXDMABURSTLENGTH_4XPBL_32BEAT ((uint32_t)0x01100000)***

maximum number of beats to be transferred in one RxDMA transaction is 32

- #define: ***ETH_RXDMABURSTLENGTH_4XPBL_64BEAT ((uint32_t)0x01200000)***

maximum number of beats to be transferred in one RxDMA transaction is 64

- #define: ***ETH_RXDMABURSTLENGTH_4XPBL_128BEAT ((uint32_t)0x01400000)***

maximum number of beats to be transferred in one RxDMA transaction is 128

ETH_Rx_Mode

- #define: ***ETH_RXPOLLING_MODE ((uint32_t)0x00000000)***
- #define: ***ETH_RXINTERRUPT_MODE ((uint32_t)0x00000001)***

ETH_Second_Frame_Operate

- #define: ***ETH_SECONDFRAMEOPERARTE_ENABLE ((uint32_t)0x00000004)***
- #define: ***ETH_SECONDFRAMEOPERARTE_DISABLE ((uint32_t)0x00000000)***

ETH_Source_Addr_Filter

- #define: ***ETH_SOURCEADDRFILTER_NORMAL_ENABLE ((uint32_t)0x00000200)***
- #define: ***ETH_SOURCEADDRFILTER_INVERSE_ENABLE ((uint32_t)0x00000300)***
- #define: ***ETH_SOURCEADDRFILTER_DISABLE ((uint32_t)0x00000000)***

ETH_Speed

- #define: ***ETH_SPEED_10M ((uint32_t)0x00000000)***
- #define: ***ETH_SPEED_100M ((uint32_t)0x00004000)***

ETH_Transmit_Flow_Control

- #define: ***ETH_TRANSMITFLOWCONTROL_ENABLE ((uint32_t)0x00000002)***

- #define: ***ETH_TRANSMITFLOWCONTROL_DISABLE ((uint32_t)0x00000000)***

ETH_Transmit_Store_Forward

- #define: ***ETH_TRANSMITSTOREFORWARD_ENABLE ((uint32_t)0x00200000)***

- #define: ***ETH_TRANSMITSTOREFORWARD_DISABLE ((uint32_t)0x00000000)***

ETH_Transmit_Threshold_Control

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_64BYTES ((uint32_t)0x00000000)***

threshold level of the MTL Transmit FIFO is 64 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_128BYTES ((uint32_t)0x00004000)***

threshold level of the MTL Transmit FIFO is 128 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_192BYTES ((uint32_t)0x00008000)***

threshold level of the MTL Transmit FIFO is 192 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_256BYTES ((uint32_t)0x0000C000)***

threshold level of the MTL Transmit FIFO is 256 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_40BYTES ((uint32_t)0x00010000)***

threshold level of the MTL Transmit FIFO is 40 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_32BYTES ((uint32_t)0x00014000)***

threshold level of the MTL Transmit FIFO is 32 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_24BYTES ((uint32_t)0x00018000)***

threshold level of the MTL Transmit FIFO is 24 Bytes

- #define: ***ETH_TRANSMITTHRESHOLDCONTROL_16BYTES ((uint32_t)0x0001C000)***

threshold level of the MTL Transmit FIFO is 16 Bytes

ETH_Tx_DMA_Burst_Length

- #define: ***ETH_TXDMABURSTLENGTH_1BEAT ((uint32_t)0x00000100)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 1

- #define: ***ETH_TXDMABURSTLENGTH_2BEAT ((uint32_t)0x00000200)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 2

- #define: ***ETH_TXDMABURSTLENGTH_4BEAT ((uint32_t)0x00000400)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 4

- #define: ***ETH_TXDMABURSTLENGTH_8BEAT ((uint32_t)0x00000800)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 8

- #define: ***ETH_TXDMABURSTLENGTH_16BEAT ((uint32_t)0x00001000)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 16

- #define: ***ETH_TXDMABURSTLENGTH_32BEAT ((uint32_t)0x00002000)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 32

- #define: ***ETH_TXDMABURSTLENGTH_4XPBL_4BEAT ((uint32_t)0x01000100)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 4

- #define: ***ETH_TXDMABURSTLENGTH_4XPBL_8BEAT ((uint32_t)0x01000200)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 8

- #define: ***ETH_TXDMABURSTLENGTH_4XPBL_16BEAT ((uint32_t)0x01000400)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 16

- #define: ***ETH_TXDMABURSTLENGTH_4XPBL_32BEAT ((uint32_t)0x01000800)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 32

- #define: ***ETH_TXDMABURSTLENGTH_4XPBL_64BEAT ((uint32_t)0x01001000)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 64

- #define: ***ETH_TXDMABURSTLENGTH_4XPBL_128BEAT ((uint32_t)0x01002000)***

maximum number of beats to be transferred in one TxDMA (or both) transaction is 128

- #define: ***ETH_MAC_ADDR_HBASE (uint32_t)(ETH_MAC_BASE + (uint32_t)0x40)***
- #define: ***ETH_MAC_ADDR_LBASE (uint32_t)(ETH_MAC_BASE + (uint32_t)0x44)***
- #define: ***MACMIIAR_CR_MASK ((uint32_t)0xFFFFFFE3)***
- #define: ***MACCR_CLEAR_MASK ((uint32_t)0xFF20810F)***
- #define: ***MACFCR_CLEAR_MASK ((uint32_t)0x0000FF41)***
- #define: ***DMAOMR_CLEAR_MASK ((uint32_t)0xF8DE3F23)***
- #define: ***ETH_WAKEUP_REGISTER_LENGTH 8***
- #define: ***ETH_DMA_RX_OVERFLOW_MISSEDFRAMES_COUNTERSHIFT 17***

ETH_Unicast_Frames_Filter

- #define: ***ETH_UNICASTFRAMESFILTER_PERFECTHASHTABLE ((uint32_t)0x00000402)***
- #define: ***ETH_UNICASTFRAMESFILTER_HASHTABLE ((uint32_t)0x00000002)***
- #define: ***ETH_UNICASTFRAMESFILTER_PERFECT ((uint32_t)0x00000000)***

ETH_Unicast_Pause_Frame_Detect

- #define: ***ETH_UNICASTPAUSEFRAMEDETECT_ENABLE ((uint32_t)0x00000008)***
- #define: ***ETH_UNICASTPAUSEFRAMEDETECT_DISABLE ((uint32_t)0x00000000)***

ETH_VLAN_Tag_Comparison

- #define: ***ETH_VLANTAGCOMPARISON_12BIT ((uint32_t)0x00010000)***
- #define: ***ETH_VLANTAGCOMPARISON_16BIT ((uint32_t)0x00000000)***

ETH_watchdog

- #define: ***ETH_WATCHDOG_ENABLE ((uint32_t)0x00000000)***
- #define: ***ETH_WATCHDOG_DISABLE ((uint32_t)0x00800000)***

ETH_Zero_Quanta_Pause

- #define: ***ETH_ZEROQUANTAPAUSE_ENABLE ((uint32_t)0x00000000)***
- #define: ***ETH_ZEROQUANTAPAUSE_DISABLE ((uint32_t)0x00000080)***

17 HAL FLASH Generic Driver

17.1 FLASH Firmware driver registers structures

17.1.1 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 FLASH_ErrorTypeDef 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 FLASH_ErrorTypeDef FLASH_ProcessTypeDef::ErrorCode***

17.1.2 FLASH_TypeDef

FLASH_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t ACR***
- ***__IO uint32_t KEYR***
- ***__IO uint32_t OPTKEYR***
- ***__IO uint32_t SR***
- ***__IO uint32_t CR***
- ***__IO uint32_t OPTCR***
- ***__IO uint32_t OPTCR1***

Field Documentation

- ***__IO uint32_t FLASH_TypeDef::ACR***
  - FLASH access control register, Address offset: 0x00
- ***__IO uint32_t FLASH_TypeDef::KEYR***
  - FLASH key register, Address offset: 0x04
- ***__IO uint32_t FLASH_TypeDef::OPTKEYR***
  - FLASH option key register, Address offset: 0x08
- ***__IO uint32_t FLASH_TypeDef::SR***
  - FLASH status register, Address offset: 0x0C
- ***__IO uint32_t FLASH_TypeDef::CR***
  - FLASH control register, Address offset: 0x10
- ***__IO uint32_t FLASH_TypeDef::OPTCR***
  - FLASH option control register , Address offset: 0x14
- ***__IO uint32_t FLASH_TypeDef::OPTCR1***
  - FLASH option control register 1, Address offset: 0x18

17.2 FLASH Firmware driver API description

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

17.2.1 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

17.2.2 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

17.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

- *HAL_FLASH_Program()*
- *HAL_FLASH_Program_IT()*
- *HAL_FLASH_IRQHandler()*
- *HAL_FLASH_EndOfOperationCallback()*
- *HAL_FLASH_OperationErrorCallback()*

17.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

- *HAL_FLASH_Unlock()*
- *HAL_FLASH_Lock()*
- *HAL_FLASH_OB_Unlock()*
- *HAL_FLASH_OB_Lock()*
- *HAL_FLASH_OB_Launch()*

17.2.5 Peripheral Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

- *HAL_FLASH_GetError()*

17.2.6 Programming operation functions

17.2.6.1 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 Status**
Notes	• None.

17.2.6.2 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**
Notes	• None.

17.2.6.3 HAL_FLASH_IRQHandler

Function Name	**void HAL_FLASH_IRQHandler (void)**
Function Description	This function handles FLASH interrupt request.
Parameters	• None.
Return values	• None.
Notes	• None.

17.2.6.4 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 0xFFFFFFFF, it means that all the selected sectors have been erased) Program: Address which was selected for data program
Return values	• None.
Notes	• None.

17.2.6.5 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.
Notes	• None.

17.2.7 Peripheral Control functions

17.2.7.1 HAL_FLASH_Unlock

Function Name	**HAL_StatusTypeDef HAL_FLASH_Unlock (void)**
Function Description	Unlock the FLASH control register access.
Parameters	• None.
Return values	• **HAL Status**

Notes	• None.

17.2.7.2 HAL_FLASH_Lock

Function Name	**HAL_StatusTypeDef HAL_FLASH_Lock (void)**
Function Description	Locks the FLASH control register access.
Parameters	• None.
Return values	• **HAL Status**
Notes	• None.

17.2.7.3 HAL_FLASH_OB_Unlock

Function Name	**HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)**
Function Description	Unlock the FLASH Option Control Registers access.
Parameters	• None.
Return values	• **HAL Status**
Notes	• None.

17.2.7.4 HAL_FLASH_OB_Lock

Function Name	**HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)**
Function Description	Lock the FLASH Option Control Registers access.
Parameters	• None.
Return values	• **HAL Status**
Notes	• None.

17.2.7.5 HAL_FLASH_OB_Launch

Function Name	**HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)**
Function Description	Launch the option byte loading.
Parameters	• None.
Return values	• **HAL Status**
Notes	• None.

17.2.8 Peripheral State and Errors functions

17.2.8.1 HAL_FLASH_GetError

Function Name	**FLASH_ErrorTypeDef HAL_FLASH_GetError (void)**
Function Description	Get the specific FLASH error flag.
Parameters	• None.
Return values	• **FLASH_ErrorCode : The returned value can be:** – ***FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP)*** – ***FLASH_ERROR_PGS: FLASH Programming Sequence error flag*** – ***FLASH_ERROR_PGP: FLASH Programming Parallelism error flag*** – ***FLASH_ERROR_PGA: FLASH Programming Alignment error flag*** – ***FLASH_ERROR_WRP: FLASH Write protected error flag*** – ***FLASH_ERROR_OPERATION: FLASH operation Error flag***
Notes	• None.

17.3 FLASH Firmware driver defines

17.3.1 FLASH

FLASH

FLASH_Exported_Constants

- #define: ***ACR_BYTE0_ADDRESS ((uint32_t)0x40023C00)***

- #define: ***OPTCR_BYTE0_ADDRESS ((uint32_t)0x40023C14)***

- #define: ***OPTCR_BYTE1_ADDRESS ((uint32_t)0x40023C15)***

- #define: ***OPTCR_BYTE2_ADDRESS ((uint32_t)0x40023C16)***

- #define: ***OPTCR_BYTE3_ADDRESS ((uint32_t)0x40023C17)***

- #define: ***OPTCR1_BYTE2_ADDRESS ((uint32_t)0x40023C1A)***

FLASH_Flag_definition

- #define: ***FLASH_FLAG_EOP FLASH_SR_EOP***

FLASH End of Operation flag

- #define: ***FLASH_FLAG_OPERR FLASH_SR_SOP***

FLASH operation Error flag

- #define: ***FLASH_FLAG_WRPERR FLASH_SR_WRPERR***

FLASH Write protected error flag

- #define: ***FLASH_FLAG_PGAERR FLASH_SR_PGAERR***

FLASH Programming Alignment error flag

- #define: ***FLASH_FLAG_PGPERR FLASH_SR_PGPERR***

FLASH Programming Parallelism error flag

- #define: ***FLASH_FLAG_PGSERR FLASH_SR_PGSERR***

FLASH Programming Sequence error flag

- #define: ***FLASH_FLAG_RDERR ((uint32_t)0x00000100)***

Read Protection error flag (PCROP)

- #define: ***FLASH_FLAG_BSY FLASH_SR_BSY***

FLASH Busy flag

FLASH_Interrupt_definition

- #define: ***FLASH_IT_EOP FLASH_CR_EOPIE***

End of FLASH Operation Interrupt source

- #define: ***FLASH_IT_ERR ((uint32_t)0x02000000)***

Error Interrupt source

FLASH_Keys

- #define: ***RDP_KEY ((uint16_t)0x00A5)***

- #define: ***FLASH_KEY1 ((uint32_t)0x45670123)***

- #define: ***FLASH_KEY2 ((uint32_t)0xCDEF89AB)***

- #define: ***FLASH_OPT_KEY1 ((uint32_t)0x08192A3B)***

- #define: ***FLASH_OPT_KEY2 ((uint32_t)0x4C5D6E7F)***

FLASH_Latency

- #define: ***FLASH_LATENCY_0 FLASH_ACR_LATENCY_0WS***

FLASH Zero Latency cycle

- #define: ***FLASH_LATENCY_1 FLASH_ACR_LATENCY_1WS***

FLASH One Latency cycle

- #define: ***FLASH_LATENCY_2 FLASH_ACR_LATENCY_2WS***

FLASH Two Latency cycles

- #define: ***FLASH_LATENCY_3 FLASH_ACR_LATENCY_3WS***

FLASH Three Latency cycles

- #define: ***FLASH_LATENCY_4 FLASH_ACR_LATENCY_4WS***

FLASH Four Latency cycles

- #define: ***FLASH_LATENCY_5 FLASH_ACR_LATENCY_5WS***

FLASH Five Latency cycles

- #define: ***FLASH_LATENCY_6 FLASH_ACR_LATENCY_6WS***

FLASH Six Latency cycles

- #define: ***FLASH_LATENCY_7 FLASH_ACR_LATENCY_7WS***

FLASH Seven Latency cycles

- #define: ***FLASH_LATENCY_8 FLASH_ACR_LATENCY_8WS***

FLASH Eight Latency cycles

- #define: ***FLASH_LATENCY_9 FLASH_ACR_LATENCY_9WS***

FLASH Nine Latency cycles

- #define: ***FLASH_LATENCY_10 FLASH_ACR_LATENCY_10WS***

FLASH Ten Latency cycles

- #define: ***FLASH_LATENCY_11 FLASH_ACR_LATENCY_11WS***

FLASH Eleven Latency cycles

- #define: ***FLASH_LATENCY_12 FLASH_ACR_LATENCY_12WS***

FLASH Twelve Latency cycles

- #define: ***FLASH_LATENCY_13 FLASH_ACR_LATENCY_13WS***

FLASH Thirteen Latency cycles

- #define: ***FLASH_LATENCY_14 FLASH_ACR_LATENCY_14WS***

FLASH Fourteen Latency cycles

- #define: ***FLASH_LATENCY_15 FLASH_ACR_LATENCY_15WS***

FLASH Fifteen Latency cycles

FLASH_Program_Parallelism

- #define: ***FLASH_PSIZE_BYTE ((uint32_t)0x00000000)***

- #define: ***FLASH_PSIZE_HALF_WORD ((uint32_t)0x00000100)***

- #define: ***FLASH_PSIZE_WORD ((uint32_t)0x00000200)***

- #define: ***FLASH_PSIZE_DOUBLE_WORD ((uint32_t)0x00000300)***

- #define: ***CR_PSIZE_MASK ((uint32_t)0xFFFFFCFF)***

FLASH_Type_Program

- #define: ***TYPEPROGRAM_BYTE ((uint32_t)0x00)***

Program byte (8-bit) at a specified address

- #define: ***TYPEPROGRAM_HALFWORD ((uint32_t)0x01)***

Program a half-word (16-bit) at a specified address

- #define: ***TYPEPROGRAM_WORD ((uint32_t)0x02)***

Program a word (32-bit) at a specified address

- #define: ***TYPEPROGRAM_DOUBLEWORD ((uint32_t)0x03)***

Program a double word (64-bit) at a specified address

18 HAL FLASH Extension Driver

18.1 FLASHEx Firmware driver registers structures

18.1.1 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***

18.1.2 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***
  - 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. This parameter can be a combination of FLASH Option Bytes IWatchdog, FLASH Option Bytes nRST_STOP and FLASH Option Bytes nRST_STDBY

18.1.3 FLASH_AdvOBProgramInitTypeDef

FLASH_AdvOBProgramInitTypeDef is defined in the stm32f4xx_hal_flash_ex.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t PCROPState***
- ***uint16_t Sectors***
- ***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***
- ***uint16_t FLASH_AdvOBProgramInitTypeDef::Sectors***

   - specifies the sector(s) set for PCROP This parameter can be a value of ***FLASHEx_Option_Bytes_PC_ReadWrite_Protection***
- ***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***

18.2 FLASHEx Firmware driver API description

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

18.2.1 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

18.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F427xx/437xx andSTM32F429xx/439xx 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 actrivation

- Write protection configuration for bank 2
- PCROP protection configuration and control for both banks

18.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extension FLASH programming operations Operations.

- *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()*

18.2.4 Extended IO operation functions

18.2.4.1 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 (0xFFFFFFFF means that all the sectors have been correctly erased)
Return values	• **HAL Status**
Notes	• None.

18.2.4.2 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**
Notes	• None.

18.2.4.3 HAL_FLASHEx_OBProgram

Function Name	**HAL_StatusTypeDef HAL_FLASHEx_OBProgram (** ***FLASH_OBProgramInitTypeDef **** **pOBInit)**
Function Description	Program option bytes.
Parameters	• **pOBInit :** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	• **HAL Status**
Notes	• None.

18.2.4.4 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.
Notes	• None.

18.2.4.5 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**
Notes	• None.

18.2.4.6 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.
Notes	• None.

18.2.4.7 HAL_FLASHEx_OB_SelectPCROP

Function Name	**HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP (void)**
Function Description	Select the Protection Mode.
Parameters	• None.
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 prtotected sector will set RDERR Flag and write a protected

sector will set WRPERR Flag
- This function can be used only for STM32F427xx/STM32F429xx/STM32F437xx/STM32F439xx/STM32F401xx devices.

18.2.4.8 HAL_FLASHEx_OB_DeSelectPCROP

Function Name	**HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP (void)**
Function Description	Deselect the Protection Mode.
Parameters	• None.
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 prtotected sector will set RDERR Flag and write a protected sector will set WRPERR Flag • This function can be used only for STM32F427xx/STM32F429xx/STM32F437xx/STM32F439xx/STM32F401xx devices.

18.2.4.9 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.
Parameters	• None.
Return values	• **The FLASH Write Protection Option Bytes value**
Notes	• This function can be used only for STM32F427X and STM32F429X devices.

18.3 FLASHEx Firmware driver defines

18.3.1 FLASHEx

FLASHEx

FLASHEx_Advanced_Option_Type

- #define: ***OBEX_PCROP ((uint32_t)0x01)***

PCROP option byte configuration

- #define: ***OBEX_BOOTCONFIG ((uint32_t)0x02)***

BOOTConfig option byte configuration

FLASHEx_Banks

- #define: ***FLASH_BANK_1 ((uint32_t)1)***

Bank 1

- #define: ***FLASH_BANK_2 ((uint32_t)2)***

Bank 2

- #define: ***FLASH_BANK_BOTH ((uint32_t)FLASH_BANK_1 | FLASH_BANK_2)***

Bank1 and Bank2

FLASHEx_BOR_Reset_Level

- #define: ***OB_BOR_LEVEL3 ((uint8_t)0x00)***

Supply voltage ranges from 2.70 to 3.60 V

- #define: ***OB_BOR_LEVEL2 ((uint8_t)0x04)***

Supply voltage ranges from 2.40 to 2.70 V

- #define: ***OB_BOR_LEVEL1 ((uint8_t)0x08)***

Supply voltage ranges from 2.10 to 2.40 V

- #define: ***OB_BOR_OFF ((uint8_t)0x0C)***

Supply voltage ranges from 1.62 to 2.10 V

FLASHEx_Dual_Boot

- #define: ***OB_DUAL_BOOT_ENABLE ((uint8_t)0x10)***

Dual Bank Boot Enable

- #define: ***OB_DUAL_BOOT_DISABLE ((uint8_t)0x00)***

Dual Bank Boot Disable, always boot on User Flash

FLASHEx_MassErase_bit

- #define: ***FLASH_MER_BIT (FLASH_CR_MER1 | FLASH_CR_MER2)***

2 MER bits here to clear

FLASHEx_Option_Bytes_IWatchdog

- #define: ***OB_IWDG_SW ((uint8_t)0x20)***

Software IWDG selected

- #define: ***OB_IWDG_HW ((uint8_t)0x00)***

Hardware IWDG selected

FLASHEx_Option_Bytes_nRST_STDBY

- #define: ***OB_STDBY_NO_RST ((uint8_t)0x80)***

No reset generated when entering in STANDBY

- #define: ***OB_STDBY_RST ((uint8_t)0x00)***

Reset generated when entering in STANDBY

FLASHEx_Option_Bytes_nRST_STOP

- #define: ***OB_STOP_NO_RST ((uint8_t)0x40)***

No reset generated when entering in STOP

- #define: ***OB_STOP_RST ((uint8_t)0x00)***

Reset generated when entering in STOP

FLASHEx_Option_Bytes_PC_ReadWrite_Protection

- #define: ***OB_PCROP_SECTOR_0 ((uint32_t)0x00000001)***

PC Read/Write protection of Sector0

- #define: ***OB_PCROP_SECTOR_1 ((uint32_t)0x00000002)***

PC Read/Write protection of Sector1

- #define: ***OB_PCROP_SECTOR_2 ((uint32_t)0x00000004)***

PC Read/Write protection of Sector2

- #define: ***OB_PCROP_SECTOR_3 ((uint32_t)0x00000008)***

PC Read/Write protection of Sector3

- #define: ***OB_PCROP_SECTOR_4 ((uint32_t)0x00000010)***

PC Read/Write protection of Sector4

- #define: ***OB_PCROP_SECTOR_5 ((uint32_t)0x00000020)***

PC Read/Write protection of Sector5

- #define: ***OB_PCROP_SECTOR_6 ((uint32_t)0x00000040)***

PC Read/Write protection of Sector6

- #define: ***OB_PCROP_SECTOR_7 ((uint32_t)0x00000080)***

PC Read/Write protection of Sector7

- #define: ***OB_PCROP_SECTOR_8 ((uint32_t)0x00000100)***

PC Read/Write protection of Sector8

- #define: ***OB_PCROP_SECTOR_9 ((uint32_t)0x00000200)***

PC Read/Write protection of Sector9

- #define: ***OB_PCROP_SECTOR_10 ((uint32_t)0x00000400)***

PC Read/Write protection of Sector10

- #define: ***OB_PCROP_SECTOR_11 ((uint32_t)0x00000800)***

PC Read/Write protection of Sector11

- #define: ***OB_PCROP_SECTOR_12 ((uint32_t)0x00000001)***

PC Read/Write protection of Sector12

- #define: ***OB_PCROP_SECTOR_13 ((uint32_t)0x00000002)***

PC Read/Write protection of Sector13

- #define: ***OB_PCROP_SECTOR_14 ((uint32_t)0x00000004)***

PC Read/Write protection of Sector14

- #define: ***OB_PCROP_SECTOR_15 ((uint32_t)0x00000008)***

PC Read/Write protection of Sector15

- #define: ***OB_PCROP_SECTOR_16 ((uint32_t)0x00000010)***

PC Read/Write protection of Sector16

- #define: ***OB_PCROP_SECTOR_17 ((uint32_t)0x00000020)***

PC Read/Write protection of Sector17

- #define: ***OB_PCROP_SECTOR_18 ((uint32_t)0x00000040)***

PC Read/Write protection of Sector18

- #define: ***OB_PCROP_SECTOR_19 ((uint32_t)0x00000080)***

PC Read/Write protection of Sector19

- #define: ***OB_PCROP_SECTOR_20 ((uint32_t)0x00000100)***

PC Read/Write protection of Sector20

- #define: ***OB_PCROP_SECTOR_21 ((uint32_t)0x00000200)***

PC Read/Write protection of Sector21

- #define: ***OB_PCROP_SECTOR_22 ((uint32_t)0x00000400)***

PC Read/Write protection of Sector22

- #define: ***OB_PCROP_SECTOR_23 ((uint32_t)0x00000800)***

PC Read/Write protection of Sector23

- #define: ***OB_PCROP_SECTOR_All ((uint32_t)0x00000FFF)***

PC Read/Write protection of all Sectors

FLASHEx_Option_Bytes_Read_Protection

- #define: ***OB_RDP_LEVEL_0 ((uint8_t)0xAA)***

- #define: ***OB_RDP_LEVEL_1 ((uint8_t)0x55)***

FLASHEx_Option_Bytes_Write_Protection

- #define: ***OB_WRP_SECTOR_0 ((uint32_t)0x00000001)***

Write protection of Sector0

- #define: ***OB_WRP_SECTOR_1 ((uint32_t)0x00000002)***

Write protection of Sector1

- #define: ***OB_WRP_SECTOR_2 ((uint32_t)0x00000004)***

Write protection of Sector2

- #define: ***OB_WRP_SECTOR_3 ((uint32_t)0x00000008)***

Write protection of Sector3

- #define: ***OB_WRP_SECTOR_4 ((uint32_t)0x00000010)***

Write protection of Sector4

- #define: ***OB_WRP_SECTOR_5 ((uint32_t)0x00000020)***

Write protection of Sector5

- #define: ***OB_WRP_SECTOR_6 ((uint32_t)0x00000040)***

Write protection of Sector6

- #define: ***OB_WRP_SECTOR_7 ((uint32_t)0x00000080)***

Write protection of Sector7

- #define: ***OB_WRP_SECTOR_8 ((uint32_t)0x00000100)***

Write protection of Sector8

- #define: ***OB_WRP_SECTOR_9 ((uint32_t)0x00000200)***

Write protection of Sector9

- #define: ***OB_WRP_SECTOR_10 ((uint32_t)0x00000400)***

Write protection of Sector10

- #define: ***OB_WRP_SECTOR_11 ((uint32_t)0x00000800)***

Write protection of Sector11

- #define: ***OB_WRP_SECTOR_12 ((uint32_t)0x00000001 << 12)***

Write protection of Sector12

- #define: ***OB_WRP_SECTOR_13 ((uint32_t)0x00000002 << 12)***

Write protection of Sector13

- #define: ***OB_WRP_SECTOR_14 ((uint32_t)0x00000004 << 12)***

Write protection of Sector14

- #define: ***OB_WRP_SECTOR_15 ((uint32_t)0x00000008 << 12)***

Write protection of Sector15

- #define: ***OB_WRP_SECTOR_16 ((uint32_t)0x00000010 << 12)***

Write protection of Sector16

- #define: ***OB_WRP_SECTOR_17 ((uint32_t)0x00000020 << 12)***

Write protection of Sector17

- #define: ***OB_WRP_SECTOR_18 ((uint32_t)0x00000040 << 12)***

Write protection of Sector18

- #define: ***OB_WRP_SECTOR_19 ((uint32_t)0x00000080 << 12)***

Write protection of Sector19

- #define: ***OB_WRP_SECTOR_20 ((uint32_t)0x00000100 << 12)***

Write protection of Sector20

- #define: ***OB_WRP_SECTOR_21 ((uint32_t)0x00000200 << 12)***

Write protection of Sector21

- #define: ***OB_WRP_SECTOR_22 ((uint32_t)0x00000400 << 12)***

Write protection of Sector22

- #define: ***OB_WRP_SECTOR_23 ((uint32_t)0x00000800 << 12)***

Write protection of Sector23

- #define: ***OB_WRP_SECTOR_All ((uint32_t)0x00000FFF << 12)***

Write protection of all Sectors

FLASHEx_Option_Type

- #define: ***OPTIONBYTE_WRP ((uint32_t)0x01)***

WRP option byte configuration

- #define: ***OPTIONBYTE_RDP ((uint32_t)0x02)***

RDP option byte configuration

- #define: ***OPTIONBYTE_USER ((uint32_t)0x04)***

USER option byte configuration

- #define: ***OPTIONBYTE_BOR ((uint32_t)0x08)***

BOR option byte configuration

FLASHEx_PCROP_State

- #define: ***PCROPSTATE_DISABLE ((uint32_t)0x00)***

Disable PCROP

- #define: ***PCROPSTATE_ENABLE ((uint32_t)0x01)***

Enable PCROP

FLASHEx_Sectors

- #define: ***FLASH_SECTOR_0 ((uint32_t)0)***

Sector Number 0

- #define: ***FLASH_SECTOR_1 ((uint32_t)1)***

Sector Number 1

- #define: ***FLASH_SECTOR_2 ((uint32_t)2)***

Sector Number 2

- #define: ***FLASH_SECTOR_3 ((uint32_t)3)***

Sector Number 3

- #define: ***FLASH_SECTOR_4 ((uint32_t)4)***

Sector Number 4

- #define: ***FLASH_SECTOR_5 ((uint32_t)5)***

Sector Number 5

- #define: ***FLASH_SECTOR_6 ((uint32_t)6)***

Sector Number 6

- #define: ***FLASH_SECTOR_7 ((uint32_t)7)***

Sector Number 7

- #define: ***FLASH_SECTOR_8 ((uint32_t)8)***

Sector Number 8

- #define: ***FLASH_SECTOR_9 ((uint32_t)9)***

Sector Number 9

- #define: ***FLASH_SECTOR_10 ((uint32_t)10)***

Sector Number 10

- #define: ***FLASH_SECTOR_11 ((uint32_t)11)***

Sector Number 11

- #define: ***FLASH_SECTOR_12 ((uint32_t)12)***

Sector Number 12

- #define: ***FLASH_SECTOR_13 ((uint32_t)13)***

Sector Number 13

- #define: ***FLASH_SECTOR_14 ((uint32_t)14)***

Sector Number 14

- #define: ***FLASH_SECTOR_15 ((uint32_t)15)***

Sector Number 15

- #define: ***FLASH_SECTOR_16 ((uint32_t)16)***

Sector Number 16

- #define: ***FLASH_SECTOR_17 ((uint32_t)17)***

Sector Number 17

- #define: ***FLASH_SECTOR_18 ((uint32_t)18)***

Sector Number 18

- #define: ***FLASH_SECTOR_19 ((uint32_t)19)***

Sector Number 19

- #define: ***FLASH_SECTOR_20 ((uint32_t)20)***

Sector Number 20

- #define: ***FLASH_SECTOR_21 ((uint32_t)21)***

Sector Number 21

- #define: ***FLASH_SECTOR_22 ((uint32_t)22)***

Sector Number 22

- #define: ***FLASH_SECTOR_23 ((uint32_t)23)***

Sector Number 23

- #define: ***FLASH_SECTOR_TOTAL 24***

FLASHEx_Selection_Protection_Mode

- #define: ***OB_PCROP_DESELECTED ((uint8_t)0x00)***

Disabled PcROP, nWPRi bits used for Write Protection on sector i

- #define: ***OB_PCROP_SELECTED ((uint8_t)0x80)***

Enable PcROP, nWPRi bits used for PCRoP Protection on sector i

FLASHEx_Type_Erase

- #define: ***TYPEERASE_SECTORS ((uint32_t)0x00)***

Sectors erase only

- #define: ***TYPEERASE_MASSERASE ((uint32_t)0x01)***

Flash Mass erase activation

FLASHEx_Voltage_Range

- #define: ***VOLTAGE_RANGE_1 ((uint32_t)0x00)***

Device operating range: 1.8V to 2.1V

- #define: ***VOLTAGE_RANGE_2 ((uint32_t)0x01)***

Device operating range: 2.1V to 2.7V

- #define: ***VOLTAGE_RANGE_3 ((uint32_t)0x02)***

Device operating range: 2.7V to 3.6V

- #define: ***VOLTAGE_RANGE_4 ((uint32_t)0x03)***

Device operating range: 2.7V to 3.6V + External Vpp

FLASHEx_WRP_State

- #define: ***WRPSTATE_DISABLE ((uint32_t)0x00)***

Disable the write protection of the desired bank 1 sectors

- #define: ***WRPSTATE_ENABLE ((uint32_t)0x01)***

Enable the write protection of the desired bank 1 sectors

19 HAL GPIO Generic Driver

19.1 GPIO Firmware driver registers structures

19.1.1 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_Alternat_function_selection***

19.1.2 GPIO_TypeDef

GPIO_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t MODER***
- ***__IO uint32_t OTYPER***
- ***__IO uint32_t OSPEEDR***
- ***__IO uint32_t PUPDR***
- ***__IO uint32_t IDR***
- ***__IO uint32_t ODR***
- ***__IO uint16_t BSRRL***
- ***__IO uint16_t BSRRH***

- ***__IO uint32_t LCKR***
- ***__IO uint32_t AFR***

Field Documentation

- ***__IO uint32_t GPIO_TypeDef::MODER***
  - GPIO port mode register, Address offset: 0x00
- ***__IO uint32_t GPIO_TypeDef::OTYPER***
  - GPIO port output type register, Address offset: 0x04
- ***__IO uint32_t GPIO_TypeDef::OSPEEDR***
  - GPIO port output speed register, Address offset: 0x08
- ***__IO uint32_t GPIO_TypeDef::PUPDR***
  - GPIO port pull-up/pull-down register, Address offset: 0x0C
- ***__IO uint32_t GPIO_TypeDef::IDR***
  - GPIO port input data register, Address offset: 0x10
- ***__IO uint32_t GPIO_TypeDef::ODR***
  - GPIO port output data register, Address offset: 0x14
- ***__IO uint16_t GPIO_TypeDef::BSRRL***
  - GPIO port bit set/reset low register, Address offset: 0x18
- ***__IO uint16_t GPIO_TypeDef::BSRRH***
  - GPIO port bit set/reset high register, Address offset: 0x1A
- ***__IO uint32_t GPIO_TypeDef::LCKR***
  - GPIO port configuration lock register, Address offset: 0x1C
- ***__IO uint32_t GPIO_TypeDef::AFR[2]***
  - GPIO alternate function registers, Address offset: 0x20-0x24

19.2 GPIO Firmware driver API description

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

19.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO

pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.

- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.
- The external interrupt/event controller consists of up to 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.

19.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __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. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
7. 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.
8. 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.

19.2.3 Initialization and de-initialization functions

- ***HAL_GPIO_Init()***
- ***HAL_GPIO_DeInit()***

19.2.4 IO operation functions

- ***HAL_GPIO_ReadPin()***
- ***HAL_GPIO_WritePin()***

- *HAL_GPIO_TogglePin()*
- *HAL_GPIO_EXTI_IRQHandler()*
- *HAL_GPIO_EXTI_Callback()*

19.2.5 Initialization and de-initialization functions

19.2.5.1 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.
Notes	• None.

19.2.5.2 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.
Notes	• None.

19.2.6 IO operation functions

19.2.6.1 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.**
Notes	• None.

19.2.6.2 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_BIT_RESET: to clear the port pin GPIO_BIT_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.

19.2.6.3 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.
Notes	• None.

19.2.6.4 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.
Notes	• None.

19.2.6.5 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.
Notes	• None.

19.3 GPIO Firmware driver defines

19.3.1 GPIO

GPIO

GPIO_Alternat_function_selection

- #define: ***GPIO_AF0_RTC_50Hz ((uint8_t)0x00)***
- #define: ***GPIO_AF0_MCO ((uint8_t)0x00)***
- #define: ***GPIO_AF0_TAMPER ((uint8_t)0x00)***
- #define: ***GPIO_AF0_SWJ ((uint8_t)0x00)***
- #define: ***GPIO_AF0_TRACE ((uint8_t)0x00)***
- #define: ***GPIO_AF1_TIM1 ((uint8_t)0x01)***
- #define: ***GPIO_AF1_TIM2 ((uint8_t)0x01)***
- #define: ***GPIO_AF2_TIM3 ((uint8_t)0x02)***
- #define: ***GPIO_AF2_TIM4 ((uint8_t)0x02)***
- #define: ***GPIO_AF2_TIM5 ((uint8_t)0x02)***
- #define: ***GPIO_AF3_TIM8 ((uint8_t)0x03)***

- #define: ***GPIO_AF3_TIM9 ((uint8_t)0x03)***
- #define: ***GPIO_AF3_TIM10 ((uint8_t)0x03)***
- #define: ***GPIO_AF3_TIM11 ((uint8_t)0x03)***
- #define: ***GPIO_AF4_I2C1 ((uint8_t)0x04)***
- #define: ***GPIO_AF4_I2C2 ((uint8_t)0x04)***
- #define: ***GPIO_AF4_I2C3 ((uint8_t)0x04)***
- #define: ***GPIO_AF5_SPI1 ((uint8_t)0x05)***
- #define: ***GPIO_AF5_SPI2 ((uint8_t)0x05)***
- #define: ***GPIO_AF5_SPI4 ((uint8_t)0x05)***
- #define: ***GPIO_AF5_SPI5 ((uint8_t)0x05)***
- #define: ***GPIO_AF5_SPI6 ((uint8_t)0x05)***
- #define: ***GPIO_AF5_I2S3ext ((uint8_t)0x05)***

- #define: ***GPIO_AF6_SPI3 ((uint8_t)0x06)***
- #define: ***GPIO_AF6_I2S2ext ((uint8_t)0x06)***
- #define: ***GPIO_AF6_SAI1 ((uint8_t)0x06)***
- #define: ***GPIO_AF7_USART1 ((uint8_t)0x07)***
- #define: ***GPIO_AF7_USART2 ((uint8_t)0x07)***
- #define: ***GPIO_AF7_USART3 ((uint8_t)0x07)***
- #define: ***GPIO_AF7_I2S3ext ((uint8_t)0x07)***
- #define: ***GPIO_AF8_UART4 ((uint8_t)0x08)***
- #define: ***GPIO_AF8_UART5 ((uint8_t)0x08)***
- #define: ***GPIO_AF8_USART6 ((uint8_t)0x08)***
- #define: ***GPIO_AF8_UART7 ((uint8_t)0x08)***
- #define: ***GPIO_AF8_UART8 ((uint8_t)0x08)***

- #define: ***GPIO_AF9_CAN1 ((uint8_t)0x09)***
- #define: ***GPIO_AF9_CAN2 ((uint8_t)0x09)***
- #define: ***GPIO_AF9_TIM12 ((uint8_t)0x09)***
- #define: ***GPIO_AF9_TIM13 ((uint8_t)0x09)***
- #define: ***GPIO_AF9_TIM14 ((uint8_t)0x09)***
- #define: ***GPIO_AF9_LTDC ((uint8_t)0x09)***
- #define: ***GPIO_AF10_OTG_FS ((uint8_t)0xA)***
- #define: ***GPIO_AF10_OTG_HS ((uint8_t)0xA)***
- #define: ***GPIO_AF11_ETH ((uint8_t)0x0B)***
- #define: ***GPIO_AF12_FMC ((uint8_t)0xC)***
- #define: ***GPIO_AF12_OTG_HS_FS ((uint8_t)0xC)***
- #define: ***GPIO_AF12_SDIO ((uint8_t)0xC)***

- #define: ***GPIO_AF13_DCMI ((uint8_t)0x0D)***

- #define: ***GPIO_AF14_LTDC ((uint8_t)0x0E)***

- #define: ***GPIO_AF15_EVENTOUT ((uint8_t)0x0F)***

GPIO_mode_define

- #define: ***GPIO_MODE_INPUT ((uint32_t)0x00000000)***

Input Floating Mode

- #define: ***GPIO_MODE_OUTPUT_PP ((uint32_t)0x00000001)***

Output Push Pull Mode

- #define: ***GPIO_MODE_OUTPUT_OD ((uint32_t)0x00000011)***

Output Open Drain Mode

- #define: ***GPIO_MODE_AF_PP ((uint32_t)0x00000002)***

Alternate Function Push Pull Mode

- #define: ***GPIO_MODE_AF_OD ((uint32_t)0x00000012)***

Alternate Function Open Drain Mode

- #define: ***GPIO_MODE_ANALOG ((uint32_t)0x00000003)***

Analog Mode

- #define: ***GPIO_MODE_IT_RISING ((uint32_t)0x10110000)***

External Interrupt Mode with Rising edge trigger detection

- #define: ***GPIO_MODE_IT_FALLING ((uint32_t)0x10210000)***

External Interrupt Mode with Falling edge trigger detection

- #define: ***GPIO_MODE_IT_RISING_FALLING ((uint32_t)0x10310000)***

External Interrupt Mode with Rising/Falling edge trigger detection

- #define: ***GPIO_MODE_EVT_RISING ((uint32_t)0x10120000)***

External Event Mode with Rising edge trigger detection

- #define: ***GPIO_MODE_EVT_FALLING ((uint32_t)0x10220000)***

External Event Mode with Falling edge trigger detection

- #define: ***GPIO_MODE_EVT_RISING_FALLING ((uint32_t)0x10320000)***

External Event Mode with Rising/Falling edge trigger detection

GPIO_pins_define

- #define: ***GPIO_PIN_0 ((uint16_t)0x0001)***
- #define: ***GPIO_PIN_1 ((uint16_t)0x0002)***
- #define: ***GPIO_PIN_2 ((uint16_t)0x0004)***
- #define: ***GPIO_PIN_3 ((uint16_t)0x0008)***
- #define: ***GPIO_PIN_4 ((uint16_t)0x0010)***
- #define: ***GPIO_PIN_5 ((uint16_t)0x0020)***
- #define: ***GPIO_PIN_6 ((uint16_t)0x0040)***
- #define: ***GPIO_PIN_7 ((uint16_t)0x0080)***

- #define: ***GPIO_PIN_8 ((uint16_t)0x0100)***

- #define: ***GPIO_PIN_9 ((uint16_t)0x0200)***

- #define: ***GPIO_PIN_10 ((uint16_t)0x0400)***

- #define: ***GPIO_PIN_11 ((uint16_t)0x0800)***

- #define: ***GPIO_PIN_12 ((uint16_t)0x1000)***

- #define: ***GPIO_PIN_13 ((uint16_t)0x2000)***

- #define: ***GPIO_PIN_14 ((uint16_t)0x4000)***

- #define: ***GPIO_PIN_15 ((uint16_t)0x8000)***

- #define: ***GPIO_PIN_All ((uint16_t)0xFFFF)***

GPIO_pull_define

- #define: ***GPIO_NOPULL ((uint32_t)0x00000000)***

No Pull-up or Pull-down activation

- #define: ***GPIO_PULLUP ((uint32_t)0x00000001)***

Pull-up activation

- #define: ***GPIO_PULLDOWN ((uint32_t)0x00000002)***

Pull-down activation

GPIO_speed_define

- #define: ***GPIO_SPEED_LOW ((uint32_t)0x00000000)***

Low speed

- #define: ***GPIO_SPEED_MEDIUM ((uint32_t)0x00000001)***

Medium speed

- #define: ***GPIO_SPEED_FAST ((uint32_t)0x00000002)***

Fast speed

- #define: ***GPIO_SPEED_HIGH ((uint32_t)0x00000003)***

High speed

20 HAL HASH Generic Driver

20.1 HASH Firmware driver registers structures

20.1.1 HASH_HandleTypeDef

HASH_HandleTypeDef is defined in the stm32f4xx_hal_hash.h

Data Fields

- ***HASH_InitTypeDef Init***
- ***uint8_t * pHashInBuffPtr***
- ***uint8_t * pHashOutBuffPtr***
- ***__IO uint32_t HashBuffSize***
- ***__IO uint32_t HashInCount***
- ***__IO uint32_t HashITCounter***
- ***HAL_StatusTypeDef Status***
- ***HAL_HASHPhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HASH_STATETypeDef State***

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 input buffer
- ***__IO uint32_t HASH_HandleTypeDef::HashBuffSize***
  - Size of buffer to be processed
- ***__IO uint32_t HASH_HandleTypeDef::HashInCount***
  - Counter of inputed data
- ***__IO uint32_t HASH_HandleTypeDef::HashITCounter***
  - Counter of issued interrupts
- ***HAL_StatusTypeDef HASH_HandleTypeDef::Status***
  - HASH peripheral status
- ***HAL_HASHPhaseTypeDef HASH_HandleTypeDef::Phase***
  - HASH peripheral phase
- ***DMA_HandleTypeDef* HASH_HandleTypeDef::hdmain***
  - HASH In DMA handle parameters
- ***HAL_LockTypeDef HASH_HandleTypeDef::Lock***
  - HASH locking object
- ***__IO HAL_HASH_STATETypeDef HASH_HandleTypeDef::State***
  - HASH peripheral state

20.1.2 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 string. 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

20.1.3 HASH_DIGEST_TypeDef

HASH_DIGEST_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t HR***

Field Documentation

- ***__IO uint32_t HASH_DIGEST_TypeDef::HR[8]***
  - HASH digest registers, Address offset: 0x310-0x32C

20.1.4 HASH_TypeDef

HASH_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t DIN***
- ***__IO uint32_t STR***
- ***__IO uint32_t HR***
- ***__IO uint32_t IMR***
- ***__IO uint32_t SR***
- ***uint32_t RESERVED***
- ***__IO uint32_t CSR***

Field Documentation

- ***__IO uint32_t HASH_TypeDef::CR***
  - HASH control register, Address offset: 0x00
- ***__IO uint32_t HASH_TypeDef::DIN***
  - HASH data input register, Address offset: 0x04
- ***__IO uint32_t HASH_TypeDef::STR***
  - HASH start register, Address offset: 0x08
- ***__IO uint32_t HASH_TypeDef::HR[5]***
  - HASH digest registers, Address offset: 0x0C-0x1C
- ***__IO uint32_t HASH_TypeDef::IMR***
  - HASH interrupt enable register, Address offset: 0x20
- ***__IO uint32_t HASH_TypeDef::SR***
  - HASH status register, Address offset: 0x24
- ***uint32_t HASH_TypeDef::RESERVED[52]***
  - Reserved, 0x28-0xF4
- ***__IO uint32_t HASH_TypeDef::CSR[54]***
  - HASH context swap registers, Address offset: 0x0F8-0x1CC

20.2 HASH Firmware driver API description

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

20.2.1 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. In case of using processing APIs based on interrupts (e.g. HAL_HMAC_SHA1_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()
   c. In case of using DMA to control data transfer (e.g. HAL_HMAC_SHA1_Start_DMA())
      - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      - Configure and enable one DMA stream one for managing 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 using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function configures mainly:
   a. The data type: 1-bit, 8-bit, 16-bit and 32-bit.
   b. For HMAC, the encryption key.
   c. For HMAC, the key size used for encryption.
3. Three processing functions 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_SHA1_Start()
   b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. HAL_HASH_SHA1_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_HASH_SHA1_Start_DMA()
4. When the processing function is called at first time after HAL_HASH_Init() the HASH peripheral is initialized and processes the buffer in input. After that, the digest computation is started. When processing multi-buffer use the accumulate function to write the data in the peripheral without starting the digest computation. In last buffer use the start function to input the last buffer ans start the digest computation.
   a. e.g. HAL_HASH_SHA1_Accumulate() : write 1st data buffer in the peripheral without starting the digest computation
   b. write (n-1)th data buffer in the peripheral without starting the digest computation
   c. HAL_HASH_SHA1_Start() : write (n)th data buffer in the peripheral and start the digest computation
5. In HMAC mode, there is no Accumulate API. Only Start API is available.
6. In case of using DMA, call the DMA start processing e.g. HAL_HASH_SHA1_Start_DMA(). After that, call the finish function in order to get the digest value e.g. HAL_HASH_SHA1_Finish()
7. Call HAL_HASH_DeInit() to deinitialize the HASH peripheral.

20.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH_InitTypeDef and creates the associated handle.
- DeInitialize the HASH peripheral.
- Initialize the HASH MSP.
- DeInitialize HASH MSP.
- ***HAL_HASH_Init()***
- ***HAL_HASH_DeInit()***
- ***HAL_HASH_MspInit()***
- ***HAL_HASH_MspDeInit()***
- ***HAL_HASH_InCpltCallback()***
- ***HAL_HASH_ErrorCallback()***
- ***HAL_HASH_DgstCpltCallback()***

20.2.3 HASH processing using polling mode functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- MD5
- SHA1
- ***HAL_HASH_MD5_Start()***
- ***HAL_HASH_MD5_Accumulate()***
- ***HAL_HASH_SHA1_Start()***
- ***HAL_HASH_SHA1_Accumulate()***

20.2.4 HASH processing using interrupt mode functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- MD5
- SHA1
- *HAL_HASH_MD5_Start_IT()*
- *HAL_HASH_SHA1_Start_IT()*
- *HAL_HASH_IRQHandler()*

20.2.5 HASH processing using DMA mode functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- MD5
- SHA1
- *HAL_HASH_MD5_Start_DMA()*
- *HAL_HASH_MD5_Finish()*
- *HAL_HASH_SHA1_Start_DMA()*
- *HAL_HASH_SHA1_Finish()*

20.2.6 HMAC processing using polling mode functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- MD5
- SHA1
- *HAL_HMAC_MD5_Start()*
- *HAL_HMAC_SHA1_Start()*

20.2.7 HMAC processing using DMA mode functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- MD5
- SHA1
- *HAL_HMAC_MD5_Start_DMA()*
- *HAL_HMAC_SHA1_Start_DMA()*

20.2.8 Peripheral State functions

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

- *HAL_HASH_GetState()*

20.2.9 Initialization and de-initialization functions

20.2.9.1 HAL_HASH_Init

Function Name	**HAL_StatusTypeDef HAL_HASH_Init (*HASH_HandleTypeDef ** hhash)**
Function Description	Initializes the HASH according to the specified parameters in the HASH_HandleTypeDef and creates the associated handle.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• **HAL status**
Notes	• None.

20.2.9.2 HAL_HASH_DeInit

Function Name	**HAL_StatusTypeDef HAL_HASH_DeInit (*HASH_HandleTypeDef ** hhash)**
Function Description	DeInitializes the HASH peripheral.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• **HAL status**
Notes	• This API must be called before starting a new processing.

20.2.9.3 HAL_HASH_MspInit

Function Name	**void HAL_HASH_MspInit (*HASH_HandleTypeDef ** hhash)**
Function Description	Initializes the HASH MSP.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.
Notes	• None.

20.2.9.4 HAL_HASH_MspDeInit

Function Name	**void HAL_HASH_MspDeInit (*HASH_HandleTypeDef ** hhash)**
Function Description	DeInitializes HASH MSP.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.
Notes	• None.

20.2.9.5 HAL_HASH_InCpltCallback

Function Name	**void HAL_HASH_InCpltCallback (*HASH_HandleTypeDef ** hhash)**
Function Description	Input data transfer complete callback.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.
Notes	• None.

20.2.9.6 HAL_HASH_ErrorCallback

Function Name	**void HAL_HASH_ErrorCallback (*HASH_HandleTypeDef ** hhash)**
Function Description	Data transfer Error callback.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.

Notes	• None.

20.2.9.7 HAL_HASH_DgstCpltCallback

Function Name	**void HAL_HASH_DgstCpltCallback (** ***HASH_HandleTypeDef **** **hhash)**
Function Description	Digest computation complete callback.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.
Notes	• This callback is not relevant with DMA.

20.2.10 HASH processing functions using polling mode

20.2.10.1 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	Initializes the HASH peripheral in MD5 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is multiple of 64 bytes, appending the input buffer is possible. If the Size is not multiple of 64 bytes, the padding is managed by hardware and appending the input buffer is no more possible. • **pOutBuffer :** Pointer to the computed digest. Its size must be 16 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

20.2.10.2 HAL_HASH_MD5_Accumulate

Function Name	**HAL_StatusTypeDef HAL_HASH_MD5_Accumulate (*HASH_HandleTypeDef ** hhash, uint8_t * pInBuffer, uint32_t Size)**
Function Description	Initializes the HASH peripheral in MD5 mode then writes the pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is multiple of 64 bytes, appending the input buffer is possible. If the Size is not multiple of 64 bytes, the padding is managed by hardware and appending the input buffer is no more possible.
Return values	• **HAL status**
Notes	• None.

20.2.10.3 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	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

20.2.10.4 HAL_HASH_SHA1_Accumulate

Function Name	**HAL_StatusTypeDef HAL_HASH_SHA1_Accumulate (*HASH_HandleTypeDef ** hhash, uint8_t * pInBuffer, uint32_t Size)**
Function Description	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes.
Return values	• **HAL status**
Notes	• None.

20.2.11 HASH processing functions using interrupt mode

20.2.11.1 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	Initializes the HASH peripheral in MD5 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pOutBuffer :** Pointer to the Output buffer (hashed buffer). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 16 bytes.
Return values	• **HAL status**
Notes	• None.

20.2.11.2 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	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes.
Return values	• **HAL status**
Notes	• None.

20.2.11.3 HAL_HASH_IRQHandler

Function Name	**void HAL_HASH_IRQHandler (*HASH_HandleTypeDef ** hhash)**
Function Description	This function handles HASH interrupt request.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.
Notes	• None.

20.2.12 HASH processing functions using DMA mode

20.2.12.1 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	Initializes the HASH peripheral in MD5 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

20.2.12.2 HAL_HASH_MD5_Finish

Function Name	**HAL_StatusTypeDef HAL_HASH_MD5_Finish (*HASH_HandleTypeDef ** hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function Description	Returns the computed digest in MD5 mode.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pOutBuffer :** Pointer to the computed digest. Its size must be 16 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

20.2.12.3 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	Initializes the HASH peripheral in SHA1 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that

	contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

20.2.12.4 HAL_HASH_SHA1_Finish

Function Name	**HAL_StatusTypeDef HAL_HASH_SHA1_Finish (*HASH_HandleTypeDef ** hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function Description	Returns the computed digest in SHA1 mode.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

20.2.13 HASH-MAC (HMAC) processing functions using polling mode

20.2.13.1 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	Initializes the HASH peripheral in HMAC MD5 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must

	be 20 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

20.2.13.2 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	Initializes the HASH peripheral in HMAC SHA1 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

20.2.14 HASH-MAC (HMAC) processing functions using DMA mode

20.2.14.1 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	Initializes the HASH peripheral in HMAC MD5 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not

	multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

20.2.14.2 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	Initializes the HASH peripheral in HMAC SHA1 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

20.2.15 Peripheral State functions

20.2.15.1 HAL_HASH_GetState

Function Name	**HAL_HASH_STATETypeDef HAL_HASH_GetState (*HASH_HandleTypeDef* * hhash)**
Function Description	return the HASH state
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• **HAL state**
Notes	• None.

20.3 HASH Firmware driver defines

20.3.1 HASH

HASH

HASH_Algorithm_Mode

- #define: ***HASH_AlgoMode_HASH ((uint32_t)0x00000000)***

Algorithm is HASH

- #define: ***HASH_AlgoMode_HMAC HASH_CR_MODE***

Algorithm is HMAC

HASH_Algo_Selection

- #define: ***HASH_AlgoSelection_SHA1 ((uint32_t)0x0000)***

HASH function is SHA1

- #define: ***HASH_AlgoSelection_SHA224 HASH_CR_ALGO_1***

HASH function is SHA224

- #define: ***HASH_AlgoSelection_SHA256 HASH_CR_ALGO***

HASH function is SHA256

- #define: ***HASH_AlgoSelection_MD5 HASH_CR_ALGO_0***

HASH function is MD5

HASH_Data_Type

- #define: ***HASH_DATATYPE_32B ((uint32_t)0x0000)***

32-bit data. No swapping

- #define: ***HASH_DATATYPE_16B HASH_CR_DATATYPE_0***

16-bit data. Each half word is swapped

- #define: ***HASH_DATATYPE_8B HASH_CR_DATATYPE_1***

8-bit data. All bytes are swapped

- #define: ***HASH_DATATYPE_1B HASH_CR_DATATYPE***

1-bit data. In the word all bits are swapped

HASH_flags_definition

- #define: ***HASH_FLAG_DINIS HASH_SR_DINIS***

16 locations are free in the DIN : A new block can be entered into the input buffer

- #define: ***HASH_FLAG_DCIS HASH_SR_DCIS***

Digest calculation complete

- #define: ***HASH_FLAG_DMAS HASH_SR_DMAS***

DMA interface is enabled (DMAE=1) or a transfer is ongoing

- #define: ***HASH_FLAG_BUSY HASH_SR_BUSY***

The hash core is Busy : processing a block of data

- #define: ***HASH_FLAG_DINNE HASH_CR_DINNE***

DIN not empty : The input buffer contains at least one word of data

HASH_HMAC_Long_key_only_for_HMAC_mode

- #define: ***HASH_HMACKeyType_ShortKey ((uint32_t)0x00000000)***

HMAC Key is <= 64 bytes

- #define: ***HASH_HMACKeyType_LongKey HASH_CR_LKEY***

HMAC Key is > 64 bytes

HASH_interrupts_definition

- #define: ***HASH_IT_DINI HASH_IMR_DINIM***

A new block can be entered into the input buffer (DIN)

- #define: ***HASH_IT_DCI HASH_IMR_DCIM***

Digest calculation complete

21 HAL HASH Extension Driver

21.1 HASHEx Firmware driver API description

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

21.1.1 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. In case of using processing APIs based on interrupts (e.g. HAL_HMACEx_SHA224_Start())
      – 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()
   c. In case of using DMA to control data transfer (e.g. HAL_HMACEx_SH224_Start_DMA())
      – Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      – Configure and enable one DMA stream one for managing 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: HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function configures mainly:
   a. The data type: 1-bit, 8-bit, 16-bit and 32-bit.
   b. For HMAC, the encryption key.
   c. For HMAC, the key size used for encryption.
3. Three processing functions 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_SHA224_Start()
   b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. HAL_HASHEx_SHA224_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_SHA224_Start_DMA()
4. When the processing function is called at first time after HAL_HASH_Init() the HASH peripheral is initialized and processes the buffer in input. After that, the digest computation is started. When processing multi-buffer use the accumulate function to write the data in the peripheral without starting the digest computation. In last buffer use the start function to input the last buffer ans start the digest computation.
   a. e.g. HAL_HASHEx_SHA224_Accumulate() : write 1st data buffer in the peripheral without starting the digest computation
   b. write (n-1)th data buffer in the peripheral without starting the digest computation
   c. HAL_HASHEx_SHA224_Start() : write (n)th data buffer in the peripheral and start the digest computation
5. In HMAC mode, there is no Accumulate API. Only Start API is available.

6. In case of using DMA, call the DMA start processing e.g. HAL_HASHEx_SHA224_Start_DMA(). After that, call the finish function in order to get the digest value e.g. HAL_HASHEx_SHA224_Finish()
7. Call HAL_HASH_DeInit() to deinitialize the HASH peripheral.

21.1.2 HASH processing using polling mode functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
- SHA256
- ***HAL_HASHEx_SHA224_Start()***
- ***HAL_HASHEx_SHA256_Start()***
- ***HAL_HASHEx_SHA224_Accumulate()***
- ***HAL_HASHEx_SHA256_Accumulate()***

21.1.3 HMAC processing using polling mode functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
- SHA256
- ***HAL_HMACEx_SHA224_Start()***
- ***HAL_HMACEx_SHA256_Start()***

21.1.4 HASH processing using interrupt functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
- SHA256
- ***HAL_HASHEx_SHA224_Start_IT()***
- ***HAL_HASHEx_SHA256_Start_IT()***
- ***HAL_HASHEx_IRQHandler()***

21.1.5 HASH processing using DMA functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
- SHA256
- ***HAL_HASHEx_SHA224_Start_DMA()***
- ***HAL_HASHEx_SHA224_Finish()***
- ***HAL_HASHEx_SHA256_Start_DMA()***
- ***HAL_HASHEx_SHA256_Finish()***

21.1.6 HMAC processing using DMA functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
- SHA256
- *HAL_HMACEx_SHA224_Start_DMA()*
- *HAL_HMACEx_SHA256_Start_DMA()*

21.1.7 HASH processing functions

21.1.7.1 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	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 28 bytes. • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

21.1.7.2 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	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not

	multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 32 bytes. • **Timeout :** Specify Timeout value
Return values	• **HAL status**
Notes	• None.

21.1.7.3 HAL_HASHEx_SHA224_Accumulate

Function Name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate (*HASH_HandleTypeDef ** hhash, uint8_t * pInBuffer, uint32_t Size)**
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

21.1.7.4 HAL_HASHEx_SHA256_Accumulate

Function Name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Accumulate (*HASH_HandleTypeDef ** hhash, uint8_t * pInBuffer, uint32_t Size)**
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**

Notes	• None.

21.1.8 HMAC processing functions using polling mode

21.1.8.1 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	Initializes the HASH peripheral in HMAC SHA224 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes.
Return values	• **HAL status**
Notes	• None.

21.1.8.2 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	Initializes the HASH peripheral in HMAC SHA256 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes.
Return values	• **HAL status**

Notes	• None.

21.1.9 HASH processing functions using interrupt mode

21.1.9.1 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	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes.
Return values	• **HAL status**
Notes	• None.

21.1.9.2 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	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware. • **pOutBuffer :** Pointer to the computed digest. Its size must be 20 bytes.
Return values	• **HAL status**

Notes	• None.

21.1.9.3 HAL_HASHEx_IRQHandler

Function Name	**void HAL_HASHEx_IRQHandler (*HASH_HandleTypeDef ** hhash)**
Function Description	This function handles HASH interrupt request.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	• None.
Notes	• None.

21.1.10 HASH processing functions using DMA mode

21.1.10.1 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	Initializes the HASH peripheral in SHA224 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

21.1.10.2 HAL_HASHEx_SHA224_Finish

Function Name	**HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (*HASH_HandleTypeDef ** hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function Description	Returns the computed digest in SHA224.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pOutBuffer :** Pointer to the computed digest. Its size must be 28 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

21.1.10.3 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	Initializes the HASH peripheral in SHA256 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

21.1.10.4 HAL_HASHEx_SHA256_Finish

Function Name	**HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (*HASH_HandleTypeDef ** hhash, uint8_t * pOutBuffer, uint32_t Timeout)**
Function Description	Returns the computed digest in SHA256.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that

	contains the configuration information for HASH module • **pOutBuffer :** Pointer to the computed digest. Its size must be 32 bytes. • **Timeout :** Timeout value
Return values	• **HAL status**
Notes	• None.

21.1.11 HMAC processing functions using DMA mode

21.1.11.1 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	Initializes the HASH peripheral in HMAC SHA224 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
Return values	• **HAL status**
Notes	• None.

21.1.11.2 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	Initializes the HASH peripheral in HMAC SHA256 mode then enables DMA to control data transfer.
Parameters	• **hhash :** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • **pInBuffer :** Pointer to the input buffer (buffer to be hashed). • **Size :** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.

Return values	• **HAL status**
Notes	• None.

21.2 HASHEx Firmware driver defines

21.2.1 HASHEx

HASHEx

22 HAL HCD Generic Driver

22.1 HCD Firmware driver registers structures

22.1.1 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***
- ***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[15]***
  – Host channels parameters
- ***HAL_LockTypeDef HCD_HandleTypeDef::Lock***
  – HCD peripheral status
- ***__IO HCD_StateTypeDef HCD_HandleTypeDef::State***
  – HCD communication state
- ***void* HCD_HandleTypeDef::pData***
  – Pointer Stack Handler

22.2 HCD Firmware driver API description

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

22.2.1 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
      – __OTGFS-OTG_CLK_ENABLE() or __OTGHS-OTG_CLK_ENABLE()

   - __OTGHSULPI_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();

22.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- *HAL_HCD_Init()*
- *HAL_HCD_HC_Init()*
- *HAL_HCD_HC_Halt()*
- *HAL_HCD_DeInit()*
- *HAL_HCD_MspInit()*
- *HAL_HCD_MspDeInit()*

22.2.3 IO operation functions

- *HAL_HCD_HC_SubmitRequest()*
- *HAL_HCD_IRQHandler()*
- *HAL_HCD_SOF_Callback()*
- *HAL_HCD_Connect_Callback()*
- *HAL_HCD_Disconnect_Callback()*
- *HAL_HCD_HC_NotifyURBChange_Callback()*

22.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

- *HAL_HCD_Start()*
- *HAL_HCD_Stop()*
- *HAL_HCD_ResetPort()*

22.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_HCD_GetState()*
- *HAL_HCD_HC_GetURBState()*
- *HAL_HCD_HC_GetXferCount()*
- *HAL_HCD_HC_GetState()*
- *HAL_HCD_GetCurrentFrame()*
- *HAL_HCD_GetCurrentSpeed()*

22.2.6 Initialization and de-initialization functions

22.2.6.1 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**
Notes	• None.

22.2.6.2 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: Isochrounous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type • **mps :** Max Packet Size. This parameter can be a value from 0 to 32K
Return values	• **HAL status**
Notes	• None.

22.2.6.3 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**
Notes	• None.

22.2.6.4 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**
Notes	• None.

22.2.6.5 HAL_HCD_MspInit

Function Name	**void HAL_HCD_MspInit (*HCD_HandleTypeDef ** hhcd)**
Function Description	Initializes the HCD MSP.
Parameters	• **hhcd :** HCD handle
Return values	• None.
Notes	• None.

22.2.6.6 HAL_HCD_MspDeInit

Function Name	**void HAL_HCD_MspDeInit (*HCD_HandleTypeDef* * hhcd)**
Function Description	DeInitializes HCD MSP.
Parameters	• **hhcd :** HCD handle
Return values	• None.
Notes	• None.

22.2.7 IO operation functions

22.2.7.1 HAL_HCD_HC_SubmitRequest

Function Name	**HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (*HCD_HandleTypeDef* * hhcd, uint8_t pipe, 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: Isochrounous 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**
Notes	• None.

22.2.7.2 HAL_HCD_IRQHandler

Function Name	**void HAL_HCD_IRQHandler (*HCD_HandleTypeDef ** hhcd)**
Function Description	This function handles HCD interrupt request.
Parameters	• **hhcd :** HCD handle
Return values	• None.
Notes	• None.

22.2.7.3 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.
Notes	• None.

22.2.7.4 HAL_HCD_Connect_Callback

Function Name	**void HAL_HCD_Connect_Callback (*HCD_HandleTypeDef ** hhcd)**
Function Description	Connexion Event callback.
Parameters	• **hhcd :** HCD handle
Return values	• None.
Notes	• None.

22.2.7.5 HAL_HCD_Disconnect_Callback

Function Name	**void HAL_HCD_Disconnect_Callback (*HCD_HandleTypeDef ** hhcd)**
Function Description	Disonnexion Event callback.
Parameters	• **hhcd :** HCD handle
Return values	• None.
Notes	• None.

22.2.7.6 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.
Notes	• None.

22.2.8 Peripheral Control functions

22.2.8.1 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**
Notes	• None.

22.2.8.2 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**
Notes	• None.

22.2.8.3 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**
Notes	• None.

22.2.9 Peripheral State functions

22.2.9.1 HAL_HCD_GetState

Function Name	**HCD_StateTypeDef HAL_HCD_GetState (*HCD_HandleTypeDef ** hhcd)**
Function Description	Return the HCD state.
Parameters	• **hhcd :** HCD handle
Return values	• **HAL state**

Notes	• None.

22.2.9.2 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**
Notes	• None.

22.2.9.3 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**
Notes	• None.

22.2.9.4 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 the these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR/**
Notes	• None.

22.2.9.5 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**
Notes	• None.

22.2.9.6 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**
Notes	• None.

22.3 HCD Firmware driver defines

22.3.1 HCD

HCD

HCD_PHY_Module

- #define: ***HCD_PHY_ULPI 1***
- #define: ***HCD_PHY_EMBEDDED 2***

HCD_Speed

- #define: ***HCD_SPEED_HIGH 0***
- #define: ***HCD_SPEED_LOW 2***
- #define: ***HCD_SPEED_FULL 3***

23 HAL I2C Generic Driver

23.1 I2C Firmware driver registers structures

23.1.1 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***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO HAL_I2C_ErrorTypeDef ErrorCode***

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
- ***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_ErrorTypeDef I2C_HandleTypeDef::ErrorCode***

23.1.2 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***

23.1.3 I2C_TypeDef

I2C_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t OAR1***
- ***__IO uint32_t OAR2***
- ***__IO uint32_t DR***
- ***__IO uint32_t SR1***

- ***__IO uint32_t SR2***
- ***__IO uint32_t CCR***
- ***__IO uint32_t TRISE***
- ***__IO uint32_t FLTR***

Field Documentation

- ***__IO uint32_t I2C_TypeDef::CR1***
  - I2C Control register 1, Address offset: 0x00
- ***__IO uint32_t I2C_TypeDef::CR2***
  - I2C Control register 2, Address offset: 0x04
- ***__IO uint32_t I2C_TypeDef::OAR1***
  - I2C Own address register 1, Address offset: 0x08
- ***__IO uint32_t I2C_TypeDef::OAR2***
  - I2C Own address register 2, Address offset: 0x0C
- ***__IO uint32_t I2C_TypeDef::DR***
  - I2C Data register, Address offset: 0x10
- ***__IO uint32_t I2C_TypeDef::SR1***
  - I2C Status register 1, Address offset: 0x14
- ***__IO uint32_t I2C_TypeDef::SR2***
  - I2C Status register 2, Address offset: 0x18
- ***__IO uint32_t I2C_TypeDef::CCR***
  - I2C Clock control register, Address offset: 0x1C
- ***__IO uint32_t I2C_TypeDef::TRISE***
  - I2C TRISE register, Address offset: 0x20
- ***__IO uint32_t I2C_TypeDef::FLTR***
  - I2C FLTR register, Address offset: 0x24

23.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

23.2.1 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 implement the 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 initilalized 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 HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customed HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function 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 HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback

- Receive in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

Interrupt mode IO MEM operation

- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At MEM end of write transfer HAL_I2C_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

DMA mode IO operation

- Transmit in master mode an amount of data in non blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At MEM end of write transfer HAL_I2C_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback

- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GET_FLAG : Checks whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG : Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

You can refer to the I2C HAL driver header file for more useful macros

23.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze 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 periperal.
- ***HAL_I2C_Init()***
- ***HAL_I2C_DeInit()***
- ***HAL_I2C_MspInit()***
- ***HAL_I2C_MspDeInit()***

23.2.3 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()
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()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2C_MemTxCpltCallback()
   - HAL_I2C_MemRxCpltCallback()
   - HAL_I2C_MasterTxCpltCallback()
   - HAL_I2C_MasterRxCpltCallback()
   - HAL_I2C_SlaveTxCpltCallback()
   - HAL_I2C_SlaveRxCpltCallback()
   - HAL_I2C_ErrorCallback()

- ***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_EV_IRQHandler()*
- *HAL_I2C_ER_IRQHandler()*
- *HAL_I2C_MasterTxCpltCallback()*
- *HAL_I2C_MasterRxCpltCallback()*
- *HAL_I2C_SlaveTxCpltCallback()*
- *HAL_I2C_SlaveRxCpltCallback()*
- *HAL_I2C_MemTxCpltCallback()*
- *HAL_I2C_MemRxCpltCallback()*
- *HAL_I2C_ErrorCallback()*

23.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_I2C_GetState()*
- *HAL_I2C_GetError()*

23.2.5 Initialization and de-initialization functions

23.2.5.1 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 create the associated handle.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• **HAL status**
Notes	• None.

23.2.5.2 HAL_I2C_DeInit

Function Name	**HAL_StatusTypeDef HAL_I2C_DeInit (** ***I2C_HandleTypeDef **** **hi2c)**
Function Description	DeInitializes the I2C peripheral.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values	• **HAL status**
Notes	• None.

23.2.5.3 HAL_I2C_MspInit

Function Name	**void HAL_I2C_MspInit (** ***I2C_HandleTypeDef **** **hi2c)**
Function Description	I2C MSP Init.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.5.4 HAL_I2C_MspDeInit

Function Name	**void HAL_I2C_MspDeInit (** ***I2C_HandleTypeDef **** **hi2c)**
Function Description	I2C MSP DeInit.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.6 IO operation functions

23.2.6.1 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 I2C module • **DevAddress :** Target device address • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

23.2.6.2 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 I2C module • **DevAddress :** Target device address • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

23.2.6.3 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 I2C module • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent

	• **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

23.2.6.4 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 I2C module • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

23.2.6.5 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 no-blocking mode with Interrupt.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **DevAddress :** Target device address • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.6 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 no-blocking mode with Interrupt.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **DevAddress :** Target device address • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.7 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 no-blocking mode with Interrupt.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.8 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 no-blocking mode with Interrupt.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.9 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 no-blocking mode with DMA.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **DevAddress :** Target device address • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.10 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 no-blocking mode with DMA.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **DevAddress :** Target device address

	• **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.11 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 no-blocking mode with DMA.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.12 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 no-blocking mode with DMA.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **pData :** Pointer to data buffer • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

23.2.6.13 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 I2C module • **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 • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

23.2.6.14 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 I2C module • **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 • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

23.2.6.15 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 no-blocking mode with Interrupt to a specific memory address.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **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**
Notes	• None.

23.2.6.16 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 no-blocking mode with Interrupt from a specific memory address.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **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**
Notes	• None.

23.2.6.17 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 no-blocking mode with DMA to a specific memory address.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **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**
Notes	• None.

23.2.6.18 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 no-blocking mode with DMA from a specific memory address.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • **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 read
Return values	• **HAL status**
Notes	• None.

23.2.6.19 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 I2C module • **DevAddress :** Target device address • **Trials :** Number of trials • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• This function is used with Memory devices

23.2.6.20 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 I2C module
Return values	• **HAL status**
Notes	• None.

23.2.6.21 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 I2C module
Return values	• **HAL status**

Notes	• None.

23.2.6.22 HAL_I2C_MasterTxCpltCallback

Function Name	**void HAL_I2C_MasterTxCpltCallback (*I2C_HandleTypeDef* * hi2c)**
Function Description	Master Tx Transfer completed callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.6.23 HAL_I2C_MasterRxCpltCallback

Function Name	**void HAL_I2C_MasterRxCpltCallback (*I2C_HandleTypeDef* * hi2c)**
Function Description	Master Rx Transfer completed callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.6.24 HAL_I2C_SlaveTxCpltCallback

Function Name	**void HAL_I2C_SlaveTxCpltCallback (*I2C_HandleTypeDef* * hi2c)**
Function Description	Slave Tx Transfer completed callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that

		contains the configuration information for I2C module
Return values	•	None.
Notes	•	None.

23.2.6.25 HAL_I2C_SlaveRxCpltCallback

Function Name	**void HAL_I2C_SlaveRxCpltCallback (*I2C_HandleTypeDef ** hi2c)**
Function Description	Slave Rx Transfer completed callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.6.26 HAL_I2C_MemTxCpltCallback

Function Name	**void HAL_I2C_MemTxCpltCallback (*I2C_HandleTypeDef ** hi2c)**
Function Description	Memory Tx Transfer completed callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.6.27 HAL_I2C_MemRxCpltCallback

Function Name	**void HAL_I2C_MemRxCpltCallback (*I2C_HandleTypeDef ** hi2c)**

Function Description	Memory Rx Transfer completed callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.6.28 HAL_I2C_ErrorCallback

Function Name	**void HAL_I2C_ErrorCallback (** ***I2C_HandleTypeDef **** **hi2c)**
Function Description	I2C error callbacks.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• None.
Notes	• None.

23.2.7 Peripheral State and Errors functions

23.2.7.1 HAL_I2C_GetState

Function Name	**HAL_I2C_StateTypeDef HAL_I2C_GetState (** ***I2C_HandleTypeDef **** **hi2c)**
Function Description	Returns the I2C state.
Parameters	• **hi2c :** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	• **HAL state**
Notes	• None.

23.2.7.2 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**
Notes	• None.

23.3 I2C Firmware driver defines

23.3.1 I2C

I2C

I2C_addressing_mode

- #define: ***I2C_ADDRESSINGMODE_7BIT ((uint32_t)0x00004000)***

- #define: ***I2C_ADDRESSINGMODE_10BIT (I2C_OAR1_ADDMODE | ((uint32_t)0x00004000))***

I2C_dual_addressing_mode

- #define: ***I2C_DUALADDRESS_DISABLED ((uint32_t)0x00000000)***

- #define: ***I2C_DUALADDRESS_ENABLED I2C_OAR2_ENDUAL***

I2C_duty_cycle_in_fast_mode

- #define: ***I2C_DUTYCYCLE_2 ((uint32_t)0x00000000)***

- #define: ***I2C_DUTYCYCLE_16_9 I2C_CCR_DUTY***

I2C_Flag_definition

- #define: ***I2C_FLAG_SMBALERT ((uint32_t)0x00018000)***
- #define: ***I2C_FLAG_TIMEOUT ((uint32_t)0x00014000)***
- #define: ***I2C_FLAG_PECERR ((uint32_t)0x00011000)***
- #define: ***I2C_FLAG_OVR ((uint32_t)0x00010800)***
- #define: ***I2C_FLAG_AF ((uint32_t)0x00010400)***
- #define: ***I2C_FLAG_ARLO ((uint32_t)0x00010200)***
- #define: ***I2C_FLAG_BERR ((uint32_t)0x00010100)***
- #define: ***I2C_FLAG_TXE ((uint32_t)0x00010080)***
- #define: ***I2C_FLAG_RXNE ((uint32_t)0x00010040)***
- #define: ***I2C_FLAG_STOPF ((uint32_t)0x00010010)***
- #define: ***I2C_FLAG_ADD10 ((uint32_t)0x00010008)***
- #define: ***I2C_FLAG_BTF ((uint32_t)0x00010004)***

- #define: ***I2C_FLAG_ADDR ((uint32_t)0x00010002)***
- #define: ***I2C_FLAG_SB ((uint32_t)0x00010001)***
- #define: ***I2C_FLAG_DUALF ((uint32_t)0x00100080)***
- #define: ***I2C_FLAG_SMBHOST ((uint32_t)0x00100040)***
- #define: ***I2C_FLAG_SMBDEFAULT ((uint32_t)0x00100020)***
- #define: ***I2C_FLAG_GENCALL ((uint32_t)0x00100010)***
- #define: ***I2C_FLAG_TRA ((uint32_t)0x00100004)***
- #define: ***I2C_FLAG_BUSY ((uint32_t)0x00100002)***
- #define: ***I2C_FLAG_MSL ((uint32_t)0x00100001)***

I2C_general_call_addressing_mode

- #define: ***I2C_GENERALCALL_DISABLED ((uint32_t)0x00000000)***
- #define: ***I2C_GENERALCALL_ENABLED I2C_CR1_ENGC***

I2C_Interrupt_configuration_definition

- #define: ***I2C_IT_BUF I2C_CR2_ITBUFEN***

- #define: ***I2C_IT_EVT I2C_CR2_ITEVTEN***

- #define: ***I2C_IT_ERR I2C_CR2_ITERREN***

I2C_Memory_Address_Size

- #define: ***I2C_MEMADD_SIZE_8BIT ((uint32_t)0x00000001)***

- #define: ***I2C_MEMADD_SIZE_16BIT ((uint32_t)0x00000010)***

I2C_nostretch_mode

- #define: ***I2C_NOSTRETCH_DISABLED ((uint32_t)0x00000000)***

- #define: ***I2C_NOSTRETCH_ENABLED I2C_CR1_NOSTRETCH***

24 HAL I2C Extension Driver

24.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

24.1.1 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

24.1.2 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()

24.1.3 Extension features functions

This section provides functions allowing to:

- Configure Noise Filters
- *HAL_I2CEx_AnalogFilter_Config()*
- *HAL_I2CEx_DigitalFilter_Config()*

24.1.4 Extension features functions

24.1.4.1 HAL_I2CEx_AnalogFilter_Config

Function Name	**HAL_StatusTypeDef HAL_I2CEx_AnalogFilter_Config (*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**
Notes	• None.

24.1.4.2 HAL_I2CEx_DigitalFilter_Config

Function Name	**HAL_StatusTypeDef HAL_I2CEx_DigitalFilter_Config (*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**
Notes	• None.

24.2 I2CEx Firmware driver defines

24.2.1 I2CEx

I2CEx

I2CEx_Analog_Filter

- #define: ***I2C_ANALOGFILTER_ENABLED ((uint32_t)0x00000000)***

- #define: ***I2C_ANALOGFILTER_DISABLED I2C_FLTR_ANOFF***

25 HAL I2S Generic Driver

25.1 I2S Firmware driver registers structures

25.1.1 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***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_I2S_StateTypeDef State***
- ***__IO HAL_I2S_ErrorTypeDef ErrorCode***

Field Documentation

- ***SPI_TypeDef* I2S_HandleTypeDef::Instance***
- ***I2S_InitTypeDef I2S_HandleTypeDef::Init***
- ***uint16_t* I2S_HandleTypeDef::pTxBuffPtr***
- ***__IO uint16_t I2S_HandleTypeDef::TxXferSize***
- ***__IO uint16_t I2S_HandleTypeDef::TxXferCount***
- ***uint16_t* I2S_HandleTypeDef::pRxBuffPtr***
- ***__IO uint16_t I2S_HandleTypeDef::RxXferSize***
- ***__IO uint16_t I2S_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx***
- ***__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock***
- ***__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State***
- ***__IO HAL_I2S_ErrorTypeDef I2S_HandleTypeDef::ErrorCode***

25.1.2 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***

25.1.3 SPI_TypeDef

SPI_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t CRCPR***
- ***__IO uint32_t RXCRCR***
- ***__IO uint32_t TXCRCR***
- ***__IO uint32_t I2SCFGR***
- ***__IO uint32_t I2SPR***

Field Documentation

- ***__IO uint32_t SPI_TypeDef::CR1***
  - SPI control register 1 (not used in I2S mode), Address offset: 0x00
- ***__IO uint32_t SPI_TypeDef::CR2***
  - SPI control register 2, Address offset: 0x04
- ***__IO uint32_t SPI_TypeDef::SR***
  - SPI status register, Address offset: 0x08
- ***__IO uint32_t SPI_TypeDef::DR***
  - SPI data register, Address offset: 0x0C
- ***__IO uint32_t SPI_TypeDef::CRCPR***
  - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10
- ***__IO uint32_t SPI_TypeDef::RXCRCR***
  - SPI RX CRC register (not used in I2S mode), Address offset: 0x14
- ***__IO uint32_t SPI_TypeDef::TXCRCR***
  - SPI TX CRC register (not used in I2S mode), Address offset: 0x18
- ***__IO uint32_t SPI_TypeDef::I2SCFGR***
  - SPI_I2S configuration register, Address offset: 0x1C
- ***__IO uint32_t SPI_TypeDef::I2SPR***
  - SPI_I2S prescaler register, Address offset: 0x20

25.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

25.2.1 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.
      - 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 initilalized 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.

3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function. The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __I2S_ENABLE_IT() and __I2S_DISABLE_IT() inside the transmit and receive process. Make sure that either: I2S 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.
4. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()

- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in USART 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

You can refer to the I2S HAL driver header file for more useful macros

25.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze 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 periperal.
- ***HAL_I2S_Init()***
- ***HAL_I2S_DeInit()***
- ***HAL_I2S_MspInit()***
- ***HAL_I2S_MspDeInit()***

25.2.3 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()

- ***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()***

25.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- ***HAL_I2S_GetState()***
- ***HAL_I2S_GetError()***

25.2.5 Initialization and de-initialization functions

25.2.5.1 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**
Notes	• None.

25.2.5.2 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**
Notes	• None.

25.2.5.3 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.
Notes	• None.

25.2.5.4 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.
Notes	• None.

25.2.6 IO operation functions

25.2.6.1 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:
Parameters	• **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).

25.2.6.2 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:
Parameters	• **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 continouse way and as the I2S is not disabled at the end of the I2S transaction.

25.2.6.3 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).

25.2.6.4 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-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.

25.2.6.5 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).

25.2.6.6 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).

25.2.6.7 HAL_I2S_DMAPause

Function Name	**HAL_StatusTypeDef HAL_I2S_DMAPause (*I2S_HandleTypeDef ** hi2s)**
Function Description	Pauses the audio stream playing from the Media.
Parameters	• **hi2s :** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL status**
Notes	• None.

25.2.6.8 HAL_I2S_DMAResume

Function Name	**HAL_StatusTypeDef HAL_I2S_DMAResume (*I2S_HandleTypeDef ** hi2s)**
Function Description	Resumes the audio stream playing from the Media.
Parameters	• **hi2s :** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values	• **HAL status**
Notes	• None.

25.2.6.9 HAL_I2S_DMAStop

Function Name	**HAL_StatusTypeDef HAL_I2S_DMAStop (** ***I2S_HandleTypeDef **** **hi2s)**
Function Description	Resumes the audio stream playing from the Media.
Parameters	• **hi2s :** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• **HAL status**
Notes	• None.

25.2.6.10 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.
Notes	• None.

25.2.6.11 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.
Notes	•	None.

25.2.6.12 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.
Notes	• None.

25.2.6.13 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.
Notes	• None.

25.2.6.14 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.
Notes	• None.

25.2.6.15 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.
Notes	• None.

25.2.7 Peripheral State and Errors functions

25.2.7.1 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**
Notes	• None.

25.2.7.2 HAL_I2S_GetError

Function Name	**HAL_I2S_ErrorTypeDef 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**
Notes	• None.

25.3 I2S Firmware driver defines

25.3.1 I2S

I2S

I2S_Audio_Frequency

- #define: ***I2S_AUDIOFREQ_192K ((uint32_t)192000)***
- #define: ***I2S_AUDIOFREQ_96K ((uint32_t)96000)***
- #define: ***I2S_AUDIOFREQ_48K ((uint32_t)48000)***
- #define: ***I2S_AUDIOFREQ_44K ((uint32_t)44100)***
- #define: ***I2S_AUDIOFREQ_32K ((uint32_t)32000)***
- #define: ***I2S_AUDIOFREQ_22K ((uint32_t)22050)***
- #define: ***I2S_AUDIOFREQ_16K ((uint32_t)16000)***
- #define: ***I2S_AUDIOFREQ_11K ((uint32_t)11025)***

- #define: ***I2S_AUDIOFREQ_8K ((uint32_t)8000)***
- #define: ***I2S_AUDIOFREQ_DEFAULT ((uint32_t)2)***

I2S_Clock_Polarity

- #define: ***I2S_CPOL_LOW ((uint32_t)0x00000000)***
- #define: ***I2S_CPOL_HIGH ((uint32_t)SPI_I2SCFGR_CKPOL)***

I2S_Clock_Source

- #define: ***I2S_CLOCK_PLL ((uint32_t)0x00000000)***
- #define: ***I2S_CLOCK_EXTERNAL ((uint32_t)0x00000001)***

I2S_Data_Format

- #define: ***I2S_DATAFORMAT_16B ((uint32_t)0x00000000)***
- #define: ***I2S_DATAFORMAT_16B_EXTENDED ((uint32_t)0x00000001)***
- #define: ***I2S_DATAFORMAT_24B ((uint32_t)0x00000003)***
- #define: ***I2S_DATAFORMAT_32B ((uint32_t)0x00000005)***

I2S_Flag_definition

- #define: ***I2S_FLAG_TXE SPI_SR_TXE***
- #define: ***I2S_FLAG_RXNE SPI_SR_RXNE***
- #define: ***I2S_FLAG_UDR SPI_SR_UDR***
- #define: ***I2S_FLAG_OVR SPI_SR_OVR***
- #define: ***I2S_FLAG_FRE SPI_SR_FRE***
- #define: ***I2S_FLAG_CHSIDE SPI_SR_CHSIDE***
- #define: ***I2S_FLAG_BSY SPI_SR_BSY***

I2S_FullDuplex_Mode

- #define: ***I2S_FULLDUPLEXMODE_DISABLE ((uint32_t)0x00000000)***
- #define: ***I2S_FULLDUPLEXMODE_ENABLE ((uint32_t)0x00000001)***

I2S_Interrupt_configuration_definition

- #define: ***I2S_IT_TXE SPI_CR2_TXEIE***
- #define: ***I2S_IT_RXNE SPI_CR2_RXNEIE***
- #define: ***I2S_IT_ERR SPI_CR2_ERRIE***

I2S_Legacy

- #define: ***I2S_STANDARD_PHILLIPS I2S_STANDARD_PHILIPS***

I2S_MCLK_Output

- #define: ***I2S_MCLKOUTPUT_ENABLE ((uint32_t)SPI_I2SPR_MCKOE)***

- #define: ***I2S_MCLKOUTPUT_DISABLE ((uint32_t)0x00000000)***

I2S_Mode

- #define: ***I2S_MODE_SLAVE_TX ((uint32_t)0x00000000)***

- #define: ***I2S_MODE_SLAVE_RX ((uint32_t)0x00000100)***

- #define: ***I2S_MODE_MASTER_TX ((uint32_t)0x00000200)***

- #define: ***I2S_MODE_MASTER_RX ((uint32_t)0x00000300)***

I2S_Standard

- #define: ***I2S_STANDARD_PHILIPS ((uint32_t)0x00000000)***

- #define: ***I2S_STANDARD_MSB ((uint32_t)0x00000010)***

- #define: ***I2S_STANDARD_LSB ((uint32_t)0x00000020)***

- #define: ***I2S_STANDARD_PCM_SHORT ((uint32_t)0x00000030)***

- #define: ***I2S_STANDARD_PCM_LONG ((uint32_t)0x000000B0)***

26 HAL I2S Extension Driver

26.1 I2SEx Firmware driver API description

The following section lists the various functions of the I2SEx library.

26.1.1 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.

Only I2Sx can deliver SCK and WS to I2Sx_ext in full duplex mode, where I2Sx can be I2S2 or I2S3.

26.1.2 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_I2S_TransmitReceive()

Interrupt mode IO operation

- Send and receive in the same time an amount of data in non blocking mode using HAL_I2S_TransmitReceive_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
- 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 and receive an amount of data in non blocking mode (DMA) using HAL_I2S_TransmitReceive_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
- 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()

26.1.3 Extension features 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_TransmitReceive()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2S_TransmitReceive_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2S_TransmitReceive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2S_TxCpltCallback()
   - HAL_I2S_RxCpltCallback()
   - HAL_I2S_ErrorCallback()

- ***HAL_I2SEx_TransmitReceive()***
- ***HAL_I2SEx_TransmitReceive_IT()***
- ***HAL_I2SEx_TransmitReceive_DMA()***

26.1.4 Extension features functions

26.1.4.1 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:
Parameters	• **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).

26.1.4.2 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).

26.1.4.3 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).

26.2 I2SEx Firmware driver defines

26.2.1 I2SEx

I2SEx

27 HAL IRDA Generic Driver

27.1 IRDA Firmware driver registers structures

27.1.1 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***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef State***
- ***__IO HAL_IRDA_ErrorTypeDef ErrorCode***

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance***
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init***
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr***
- ***uint16_t IRDA_HandleTypeDef::TxXferSize***
- ***uint16_t IRDA_HandleTypeDef::TxXferCount***
- ***uint8_t* IRDA_HandleTypeDef::pRxBuffPtr***
- ***uint16_t IRDA_HandleTypeDef::RxXferSize***
- ***uint16_t IRDA_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::State***
- ***__IO HAL_IRDA_ErrorTypeDef IRDA_HandleTypeDef::ErrorCode***

27.1.2 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***
- ***uint32_t IRDA_InitTypeDef::Mode***
  - Specifies wether 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
- ***uint32_t IRDA_InitTypeDef::IrDAMode***
  - Specifies the IrDA mode This parameter can be a value of ***IrDA_Low_Power***

27.1.3 USART_TypeDef

USART_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t BRR***
- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t CR3***
- ***__IO uint32_t GTPR***

Field Documentation

- ***__IO uint32_t USART_TypeDef::SR***
  - USART Status register, Address offset: 0x00
- ***__IO uint32_t USART_TypeDef::DR***
  - USART Data register, Address offset: 0x04
- ***__IO uint32_t USART_TypeDef::BRR***
  - USART Baud rate register, Address offset: 0x08

- ***__IO uint32_t USART_TypeDef::CR1***
  – USART Control register 1, Address offset: 0x0C
- ***__IO uint32_t USART_TypeDef::CR2***
  – USART Control register 2, Address offset: 0x10
- ***__IO uint32_t USART_TypeDef::CR3***
  – USART Control register 3, Address offset: 0x14
- ***__IO uint32_t USART_TypeDef::GTPR***
  – USART Guard time and prescaler register, Address offset: 0x18

27.2 IRDA Firmware driver API description

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

27.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure.
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 these 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 initilalized 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.
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 customed HAL_IRDA_MspInit() API. 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 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 transfer HAL_IRDA_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback

IRDA HAL driver macros list

You can refer to the IRDA HAL driver header file for more useful macros

27.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in 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. Refer to STM32F4xx reference manual (RM0090) for the IRDA frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - 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).

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***

27.2.3 IO operation functions

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.
   - 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 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 respectivelly 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 API's 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()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_IRDA_TxCpltCallback()
   - HAL_IRDA_RxCpltCallback()
   - HAL_IRDA_ErrorCallback()

- ***HAL_IRDA_Transmit()***
- ***HAL_IRDA_Receive()***
- ***HAL_IRDA_Transmit_IT()***
- ***HAL_IRDA_Receive_IT()***
- ***HAL_IRDA_Transmit_DMA()***
- ***HAL_IRDA_Receive_DMA()***
- ***HAL_IRDA_IRQHandler()***
- ***HAL_IRDA_TxCpltCallback()***
- ***HAL_IRDA_RxCpltCallback()***
- ***HAL_IRDA_ErrorCallback()***

27.2.4 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.
- ***HAL_IRDA_GetState()***
- ***HAL_IRDA_GetError()***

27.2.5 IrDA Initialization and de-initialization functions

27.2.5.1 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**
Notes	• None.

27.2.5.2 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**
Notes	• None.

27.2.5.3 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.
Notes	• None.

27.2.5.4 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.
Notes	• None.

27.2.6 IO operation functions

27.2.6.1 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 • **Size :** Amount of data to be sent

	• **Timeout :** Specify timeout value
Return values	• **HAL status**
Notes	• None.

27.2.6.2 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 • **Size :** Amount of data to be received • **Timeout :** Specify timeout value
Return values	• **HAL status**
Notes	• None.

27.2.6.3 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 • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

27.2.6.4 HAL_IRDA_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_IRDA_Receive_IT (*IRDA_HandleTypeDef ** hirda, uint8_t * pData, uint16_t Size)**
Function Description	Receives 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 • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• None.

27.2.6.5 HAL_IRDA_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (*IRDA_HandleTypeDef ** hirda, uint8_t * pData, uint16_t Size)**
Function Description	Sends 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 • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

27.2.6.6 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 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 • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• When the IRDA parity is enabled (PCE = 1) the data received contain the parity bit.

27.2.6.7 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.
Notes	• None.

27.2.6.8 HAL_IRDA_TxCpltCallback

Function Name	**void HAL_IRDA_TxCpltCallback (*IRDA_HandleTypeDef ** hirda)**
Function Description	Tx Transfer complete callbacks.
Parameters	• **hirda :** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None.
Notes	• None.

27.2.6.9 HAL_IRDA_RxCpltCallback

Function Name	**void HAL_IRDA_RxCpltCallback (** ***IRDA_HandleTypeDef **** **hirda)**
Function Description	Rx Transfer complete callbacks.
Parameters	• **hirda :** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None.
Notes	• None.

27.2.6.10 HAL_IRDA_ErrorCallback

Function Name	**void HAL_IRDA_ErrorCallback (** ***IRDA_HandleTypeDef **** **hirda)**
Function Description	IRDA error callbacks.
Parameters	• **hirda :** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None.
Notes	• None.

27.2.7 Peripheral State and Errors functions

27.2.7.1 HAL_IRDA_GetState

Function Name	**HAL_IRDA_StateTypeDef HAL_IRDA_GetState (** ***IRDA_HandleTypeDef **** **hirda)**
Function Description	Returns the IRDA state.
Parameters	• **hirda :** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• **HAL state**
Notes	• None.

27.2.7.2 HAL_IRDA_GetError

Function Name	**uint32_t HAL_IRDA_GetError (*IRDA_HandleTypeDef ** hirda)**
Function Description	Return the IARDA error code.
Parameters	• **hirda :** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA.
Return values	• **IRDA Error Code**
Notes	• None.

27.3 IRDA Firmware driver defines

27.3.1 IRDA

IRDA

IRDA_Flags

- #define: ***IRDA_FLAG_TXE ((uint32_t)0x00000080)***
- #define: ***IRDA_FLAG_TC ((uint32_t)0x00000040)***
- #define: ***IRDA_FLAG_RXNE ((uint32_t)0x00000020)***
- #define: ***IRDA_FLAG_IDLE ((uint32_t)0x00000010)***
- #define: ***IRDA_FLAG_ORE ((uint32_t)0x00000008)***
- #define: ***IRDA_FLAG_NE ((uint32_t)0x00000004)***

- #define: ***IRDA_FLAG_FE ((uint32_t)0x00000002)***

- #define: ***IRDA_FLAG_PE ((uint32_t)0x00000001)***

IRDA_Interrupt_definition

- #define: ***IRDA_IT_PE ((uint32_t)0x10000100)***

- #define: ***IRDA_IT_TXE ((uint32_t)0x10000080)***

- #define: ***IRDA_IT_TC ((uint32_t)0x10000040)***

- #define: ***IRDA_IT_RXNE ((uint32_t)0x10000020)***

- #define: ***IRDA_IT_IDLE ((uint32_t)0x10000010)***

- #define: ***IRDA_IT_LBD ((uint32_t)0x20000040)***

- #define: ***IRDA_IT_CTS ((uint32_t)0x30000400)***

- #define: ***IRDA_IT_ERR ((uint32_t)0x30000001)***

IrDA_Low_Power

- #define: ***IRDA_POWERMODE_LOWPOWER ((uint32_t)USART_CR3_IRLP)***

- #define: ***IRDA_POWERMODE_NORMAL ((uint32_t)0x00000000)***

IRDA_Mode

- #define: ***IRDA_MODE_RX ((uint32_t)USART_CR1_RE)***
- #define: ***IRDA_MODE_TX ((uint32_t)USART_CR1_TE)***
- #define: ***IRDA_MODE_TX_RX ((uint32_t)(USART_CR1_TE |USART_CR1_RE))***

IRDA_Parity

- #define: ***IRDA_PARITY_NONE ((uint32_t)0x00000000)***
- #define: ***IRDA_PARITY_EVEN ((uint32_t)USART_CR1_PCE)***
- #define: ***IRDA_PARITY_ODD ((uint32_t)(USART_CR1_PCE | USART_CR1_PS))***

IRDA_Word_Length

- #define: ***IRDA_WORDLENGTH_8B ((uint32_t)0x00000000)***
- #define: ***IRDA_WORDLENGTH_9B ((uint32_t)USART_CR1_M)***

28 HAL IWDG Generic Driver

28.1 IWDG Firmware driver registers structures

28.1.1 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32f4xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IWDG_StateTypeDef State***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
  - Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
  - IWDG required parameters
- ***HAL_LockTypeDef IWDG_HandleTypeDef::Lock***
  - IWDG locking object
- ***__IO HAL_IWDG_StateTypeDef IWDG_HandleTypeDef::State***
  - IWDG communication state

28.1.2 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

28.1.3 IWDG_TypeDef

IWDG_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t KR***
- ***__IO uint32_t PR***
- ***__IO uint32_t RLR***
- ***__IO uint32_t SR***

Field Documentation

- ***__IO uint32_t IWDG_TypeDef::KR***
  - IWDG Key register, Address offset: 0x00
- ***__IO uint32_t IWDG_TypeDef::PR***
  - IWDG Prescaler register, Address offset: 0x04
- ***__IO uint32_t IWDG_TypeDef::RLR***
  - IWDG Reload register, Address offset: 0x08
- ***__IO uint32_t IWDG_TypeDef::SR***
  - IWDG Status register, Address offset: 0x0C

28.2 IWDG Firmware driver API description

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

28.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by its own dedicated 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 cannot be disabled (LSI cannot be disabled too), and the counter starts counting down from the reset value of 0xFFF. When it reaches the end of count value (0x000) a system reset is generated.
- The IWDG counter should be refreshed at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0.
- 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.
- 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.

28.2.2 How to use this driver

- Use IWDG using HAL_IWDG_Start() function to:
  – Enable write access to IWDG_PR and IWDG_RLR registers.
  – Configure the IWDG prescaler and counter reload values.
  – Reload IWDG counter with value defined in the IWDG_RLR register.
  – Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
- 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
- __HAL_IWDG_ENABLE_WRITE_ACCESS : Enable write access to IWDG_PR and IWDG_RLR registers
- __HAL_IWDG_DISABLE_WRITE_ACCESS : Disable write access to IWDG_PR and IWDG_RLR registers
- __HAL_IWDG_GET_FLAG: Get the selected IWDG's flag status
- __HAL_IWDG_CLEAR_FLAG: Clear the IWDG's pending flags

28.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and create the associated handle
- Initialize the IWDG MSP
- DeInitialize IWDG MSP
- ***HAL_IWDG_Init()***
- ***HAL_IWDG_MspInit()***

28.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.
- ***HAL_IWDG_Start()***
- ***HAL_IWDG_Refresh()***

28.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_IWDG_GetState()*

28.2.6 Initialization and de-initialization functions

28.2.6.1 HAL_IWDG_Init

Function Name	**HAL_StatusTypeDef HAL_IWDG_Init (** ***IWDG_HandleTypeDef **** **hiwdg)**
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
Parameters	• **hiwdg :** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL status**
Notes	• None.

28.2.6.2 HAL_IWDG_MspInit

Function Name	**void HAL_IWDG_MspInit (** ***IWDG_HandleTypeDef **** **hiwdg)**
Function Description	Initializes the IWDG MSP.
Parameters	• **hiwdg :** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• None.
Notes	• None.

28.2.7 IO operation functions

28.2.7.1 HAL_IWDG_Start

Function Name	**HAL_StatusTypeDef HAL_IWDG_Start (** ***IWDG_HandleTypeDef **** **hiwdg)**
Function Description	Starts the IWDG.

Parameters	• **hiwdg :** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL status**
Notes	• None.

28.2.7.2 HAL_IWDG_Refresh

Function Name	**HAL_StatusTypeDef HAL_IWDG_Refresh (*IWDG_HandleTypeDef ** hiwdg)**
Function Description	Refreshes the IWDG.
Parameters	• **hiwdg :** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL status**
Notes	• None.

28.2.8 Peripheral State functions

28.2.8.1 HAL_IWDG_GetState

Function Name	**HAL_IWDG_StateTypeDef HAL_IWDG_GetState (*IWDG_HandleTypeDef ** hiwdg)**
Function Description	Returns the IWDG state.
Parameters	• **hiwdg :** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• **HAL state**
Notes	• None.

28.3 IWDG Firmware driver defines

28.3.1 IWDG

IWDG

IWDG_Flag_definition

- #define: ***IWDG_FLAG_PVU ((uint32_t)0x0001)***

Watchdog counter prescaler value update flag

- #define: ***IWDG_FLAG_RVU ((uint32_t)0x0002)***

Watchdog counter reload value update flag

IWDG_Prescaler

- #define: ***IWDG_PRESCALER_4 ((uint8_t)0x00)***

IWDG prescaler set to 4

- #define: ***IWDG_PRESCALER_8 ((uint8_t)0x01)***

IWDG prescaler set to 8

- #define: ***IWDG_PRESCALER_16 ((uint8_t)0x02)***

IWDG prescaler set to 16

- #define: ***IWDG_PRESCALER_32 ((uint8_t)0x03)***

IWDG prescaler set to 32

- #define: ***IWDG_PRESCALER_64 ((uint8_t)0x04)***

IWDG prescaler set to 64

- #define: ***IWDG_PRESCALER_128 ((uint8_t)0x05)***

IWDG prescaler set to 128

- #define: ***IWDG_PRESCALER_256 ((uint8_t)0x06)***

IWDG prescaler set to 256

IWDG_Registers_BitMask

- #define: ***KR_KEY_RELOAD ((uint32_t)0xAAAA)***

IWDG reload counter enable

- #define: ***KR_KEY_ENABLE ((uint32_t)0xCCCC)***

IWDG peripheral enable

- #define: ***KR_KEY_EWA ((uint32_t)0x5555)***

IWDG KR write Access enable

- #define: ***KR_KEY_DWA ((uint32_t)0x0000)***

IWDG KR write Access disable

29 HAL LTDC Generic Driver

29.1 LTDC Firmware driver registers structures

29.1.1 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

29.1.2 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 heigh. 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 heigh. 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 heigh. 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 heigh. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveH and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_InitTypeDef::Backcolor***
  – Configures the background color.

29.1.3 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

29.1.4 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 = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY1***
  - Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***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.

29.1.5 LTDC_Layer_TypeDef

LTDC_Layer_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t WHPCR***
- ***__IO uint32_t WVPCR***
- ***__IO uint32_t CKCR***
- ***__IO uint32_t PFCR***
- ***__IO uint32_t CACR***
- ***__IO uint32_t DCCR***
- ***__IO uint32_t BFCR***
- ***uint32_t RESERVED0***
- ***__IO uint32_t CFBAR***
- ***__IO uint32_t CFBLR***
- ***__IO uint32_t CFBLNR***

- ***uint32_t RESERVED1***
- ***__IO uint32_t CLUTWR***

Field Documentation

- ***__IO uint32_t LTDC_Layer_TypeDef::CR***
  - LTDC Layerx Control Register Address offset: 0x84
- ***__IO uint32_t LTDC_Layer_TypeDef::WHPCR***
  - LTDC Layerx Window Horizontal Position Configuration Register Address offset: 0x88
- ***__IO uint32_t LTDC_Layer_TypeDef::WVPCR***
  - LTDC Layerx Window Vertical Position Configuration Register Address offset: 0x8C
- ***__IO uint32_t LTDC_Layer_TypeDef::CKCR***
  - LTDC Layerx Color Keying Configuration Register Address offset: 0x90
- ***__IO uint32_t LTDC_Layer_TypeDef::PFCR***
  - LTDC Layerx Pixel Format Configuration Register Address offset: 0x94
- ***__IO uint32_t LTDC_Layer_TypeDef::CACR***
  - LTDC Layerx Constant Alpha Configuration Register Address offset: 0x98
- ***__IO uint32_t LTDC_Layer_TypeDef::DCCR***
  - LTDC Layerx Default Color Configuration Register Address offset: 0x9C
- ***__IO uint32_t LTDC_Layer_TypeDef::BFCR***
  - LTDC Layerx Blending Factors Configuration Register Address offset: 0xA0
- ***uint32_t LTDC_Layer_TypeDef::RESERVED0[2]***
  - Reserved
- ***__IO uint32_t LTDC_Layer_TypeDef::CFBAR***
  - LTDC Layerx Color Frame Buffer Address Register Address offset: 0xAC
- ***__IO uint32_t LTDC_Layer_TypeDef::CFBLR***
  - LTDC Layerx Color Frame Buffer Length Register Address offset: 0xB0
- ***__IO uint32_t LTDC_Layer_TypeDef::CFBLNR***
  - LTDC Layerx ColorFrame Buffer Line Number Register Address offset: 0xB4
- ***uint32_t LTDC_Layer_TypeDef::RESERVED1[3]***
  - Reserved
- ***__IO uint32_t LTDC_Layer_TypeDef::CLUTWR***
  - LTDC Layerx CLUT Write Register Address offset: 0x144

29.1.6 LTDC_TypeDef

LTDC_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***uint32_t RESERVED0***
- ***__IO uint32_t SSCR***
- ***__IO uint32_t BPCR***
- ***__IO uint32_t AWCR***
- ***__IO uint32_t TWCR***
- ***__IO uint32_t GCR***
- ***uint32_t RESERVED1***
- ***__IO uint32_t SRCR***

- ***uint32_t RESERVED2***
- ***__IO uint32_t BCCR***
- ***uint32_t RESERVED3***
- ***__IO uint32_t IER***
- ***__IO uint32_t ISR***
- ***__IO uint32_t ICR***
- ***__IO uint32_t LIPCR***
- ***__IO uint32_t CPSR***
- ***__IO uint32_t CDSR***

Field Documentation

- ***uint32_t LTDC_TypeDef::RESERVED0[2]***
  - Reserved, 0x00-0x04
- ***__IO uint32_t LTDC_TypeDef::SSCR***
  - LTDC Synchronization Size Configuration Register, Address offset: 0x08
- ***__IO uint32_t LTDC_TypeDef::BPCR***
  - LTDC Back Porch Configuration Register, Address offset: 0x0C
- ***__IO uint32_t LTDC_TypeDef::AWCR***
  - LTDC Active Width Configuration Register, Address offset: 0x10
- ***__IO uint32_t LTDC_TypeDef::TWCR***
  - LTDC Total Width Configuration Register, Address offset: 0x14
- ***__IO uint32_t LTDC_TypeDef::GCR***
  - LTDC Global Control Register, Address offset: 0x18
- ***uint32_t LTDC_TypeDef::RESERVED1[2]***
  - Reserved, 0x1C-0x20
- ***__IO uint32_t LTDC_TypeDef::SRCR***
  - LTDC Shadow Reload Configuration Register, Address offset: 0x24
- ***uint32_t LTDC_TypeDef::RESERVED2[1]***
  - Reserved, 0x28
- ***__IO uint32_t LTDC_TypeDef::BCCR***
  - LTDC Background Color Configuration Register, Address offset: 0x2C
- ***uint32_t LTDC_TypeDef::RESERVED3[1]***
  - Reserved, 0x30
- ***__IO uint32_t LTDC_TypeDef::IER***
  - LTDC Interrupt Enable Register, Address offset: 0x34
- ***__IO uint32_t LTDC_TypeDef::ISR***
  - LTDC Interrupt Status Register, Address offset: 0x38
- ***__IO uint32_t LTDC_TypeDef::ICR***
  - LTDC Interrupt Clear Register, Address offset: 0x3C
- ***__IO uint32_t LTDC_TypeDef::LIPCR***
  - LTDC Line Interrupt Position Configuration Register, Address offset: 0x40
- ***__IO uint32_t LTDC_TypeDef::CPSR***
  - LTDC Current Position Status Register, Address offset: 0x44
- ***__IO uint32_t LTDC_TypeDef::CDSR***
  - LTDC Current Display Status Register, Address offset: 0x48

29.2 LTDC Firmware driver API description

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

29.2.1 How to use this driver

1. Program 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
2. 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.
3. Optionally, configure and enable the CLUT using HAL_LTDC_ConfigCLUT() and HAL_LTDC_EnableCLUT functions.
4. Optionally, enable the Dither using HAL_LTDC_EnableDither().
5. Optionally, configure and enable the Color keying using HAL_LTDC_ConfigColorKeying() and HAL_LTDC_EnableColorKeying functions.
6. Optionally, configure LineInterrupt using HAL_LTDC_ProgramLineInterrupt() function
7. 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(), HAL_LTDC_SetAddress.
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 the LTDC Layer.
- __HAL_LTDC_LAYER_DISABLE: Disable the LTDC Layer.
- __HAL_LTDC_RELOAD_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.

You can refer to the LTDC HAL driver header file for more useful macros

29.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC

- De-initialize the LTDC
- *HAL_LTDC_Init()*
- *HAL_LTDC_DeInit()*
- *HAL_LTDC_MspInit()*
- *HAL_LTDC_MspDeInit()*

29.2.3 IO operation functions

This section provides function allowing to:

- handle LTDC interrupt request
- *HAL_LTDC_IRQHandler()*
- *HAL_LTDC_ErrorCallback()*
- *HAL_LTDC_LineEvenCallback()*

29.2.4 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.
- *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_ProgramLineEvent()*

29.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC state.

- Get error code.
- *HAL_LTDC_GetState()*
- *HAL_LTDC_GetError()*

29.2.6 Initialization and Configuration functions

29.2.6.1 HAL_LTDC_Init

Function Name	**HAL_StatusTypeDef HAL_LTDC_Init (** ***LTDC_HandleTypeDef**** **hltdc)**
Function Description	Initializes the LTDC 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.
Return values	• **HAL status**
Notes	• None.

29.2.6.2 HAL_LTDC_DeInit

Function Name	**HAL_StatusTypeDef HAL_LTDC_DeInit (** ***LTDC_HandleTypeDef**** **hltdc)**
Function Description	Deinitializes the LTDC peripheral registers to their default reset values.
Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• None.
Notes	• None.

29.2.6.3 HAL_LTDC_MspInit

Function Name	**void HAL_LTDC_MspInit (** ***LTDC_HandleTypeDef**** **hltdc)**
Function Description	Initializes the LTDC MSP.

Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• None.
Notes	• None.

29.2.6.4 HAL_LTDC_MspDeInit

Function Name	**void HAL_LTDC_MspDeInit (** ***LTDC_HandleTypeDef ** **hltdc)**
Function Description	DeInitializes the LTDC MSP.
Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• None.
Notes	• None.

29.2.7 IO operation functions

29.2.7.1 HAL_LTDC_IRQHandler

Function Name	**void HAL_LTDC_IRQHandler (** ***LTDC_HandleTypeDef ** **hltdc)**
Function Description	Handles LTDC interrupt request.
Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL status**
Notes	• None.

29.2.7.2 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.
Notes	• None.

29.2.7.3 HAL_LTDC_LineEvenCallback

Function Name	**void HAL_LTDC_LineEvenCallback (*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.
Notes	• None.

29.2.8 Peripheral Control functions

29.2.8.1 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.2 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.3 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.4 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.5 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.6 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: 0 or 1
Return values	• **HAL status**

Notes	• None.

29.2.8.7 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.8 HAL_LTDC_EnableDither

Function Name	**HAL_StatusTypeDef HAL_LTDC_EnableDither (** ***LTDC_HandleTypeDef* *** **hltdc)**
Function Description	Enables Dither.
Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL status**
Notes	• None.

29.2.8.9 HAL_LTDC_DisableDither

Function Name	**HAL_StatusTypeDef HAL_LTDC_DisableDither (** ***LTDC_HandleTypeDef* *** **hltdc)**
Function Description	Disables Dither.

Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL status**
Notes	• None.

29.2.8.10 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.11 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.12 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: 0 or 1.
Return values	• **HAL status**
Notes	• None.

29.2.8.13 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: 0 or 1
Return values	• **HAL status**
Notes	• None.

29.2.8.14 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: 0 or 1.
Return values	• **HAL status**
Notes	• None.

29.2.8.15 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	• None.

29.2.9 Peripheral State and Errors functions

29.2.9.1 HAL_LTDC_GetState

Function Name	**HAL_LTDC_StateTypeDef HAL_LTDC_GetState (*LTDC_HandleTypeDef ** hltdc)**
Function Description	Return the LTDC state.
Parameters	• **hltdc :** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **HAL state**
Notes	• None.

29.2.9.2 HAL_LTDC_GetError

Function Name	**uint32_t HAL_LTDC_GetError (*LTDC_HandleTypeDef ** hltdc)**
Function Description	Return the LTDC error code.
Parameters	• **hltdc :** : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	• **LTDC Error Code**
Notes	• None.

29.3 LTDC Firmware driver defines

29.3.1 LTDC

LTDC

LTDC_Alpha

- #define: ***LTDC_ALPHA LTDC_LxCACR_CONSTA***

LTDC Cte Alpha mask

LTDC_BACK_COLOR

- #define: ***LTDC_COLOR ((uint32_t)0x000000FF)***

Color mask

LTDC_BlendingFactor1

- #define: ***LTDC_BLENDING_FACTOR1_CA ((uint32_t)0x00000400)***

Blending factor : Cte Alpha

- #define: ***LTDC_BLENDING_FACTOR1_PAxCA ((uint32_t)0x00000600)***

Blending factor : Cte Alpha x Pixel Alpha

LTDC_BlendingFactor2

- #define: ***LTDC_BLENDING_FACTOR2_CA ((uint32_t)0x00000005)***

Blending factor : Cte Alpha

- #define: ***LTDC_BLENDING_FACTOR2_PAxCA ((uint32_t)0x00000007)***

Blending factor : Cte Alpha x Pixel Alpha

LTDC_DE_POLARITY

- #define: ***LTDC_DEPOLARITY_AL ((uint32_t)0x00000000)***

Data Enable, is active low.

- #define: ***LTDC_DEPOLARITY_AH LTDC_GCR_DEPOL***

Data Enable, is active high.

LTDC_Flag

- #define: ***LTDC_FLAG_LI LTDC_ISR_LIF***

- #define: ***LTDC_FLAG_FU LTDC_ISR_FUIF***

- #define: ***LTDC_FLAG_TE LTDC_ISR_TERRIF***

- #define: ***LTDC_FLAG_RR LTDC_ISR_RRIF***

LTDC_HS_POLARITY

- #define: ***LTDC_HSPOLARITY_AL ((uint32_t)0x00000000)***

Horizontal Synchronization is active low.

- #define: ***LTDC_HSPOLARITY_AH LTDC_GCR_HSPOL***

Horizontal Synchronization is active high.

LTDC_Interrupts

- #define: ***LTDC_IT_LI LTDC_IER_LIE***

- #define: ***LTDC_IT_FU LTDC_IER_FUIE***

- #define: ***LTDC_IT_TE LTDC_IER_TERRIE***

- #define: ***LTDC_IT_RR LTDC_IER_RRIE***

LTDC_LAYER_Config

- #define: ***LTDC_STOPPOSITION (LTDC_LxWHPCR_WHSPPOS >> 16)***

LTDC Layer stop position

- #define: ***LTDC_STARTPOSITION LTDC_LxWHPCR_WHSTPOS***

LTDC Layer start position

- #define: ***LTDC_COLOR_FRAME_BUFFER LTDC_LxCFBLR_CFBLL***

LTDC Layer Line length

- #define: ***LTDC_LINE_NUMBER LTDC_LxCFBLNR_CFBLNBR***

LTDC Layer Line number

LTDC_PC_POLARITY

- #define: ***LTDC_PCPOLARITY_IPC ((uint32_t)0x00000000)***

input pixel clock.

- #define: ***LTDC_PCPOLARITY_IIPC LTDC_GCR_PCPOL***

inverted input pixel clock.

LTDC_Pixelformat

- #define: ***LTDC_PIXEL_FORMAT_ARGB8888 ((uint32_t)0x00000000)***

ARGB8888 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_RGB888 ((uint32_t)0x00000001)***

RGB888 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_RGB565 ((uint32_t)0x00000002)***

RGB565 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_ARGB1555 ((uint32_t)0x00000003)***

ARGB1555 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_ARGB4444 ((uint32_t)0x00000004)***

ARGB4444 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_L8 ((uint32_t)0x00000005)***

L8 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_AL44 ((uint32_t)0x00000006)***

AL44 LTDC pixel format

- #define: ***LTDC_PIXEL_FORMAT_AL88 ((uint32_t)0x00000007)***

AL88 LTDC pixel format

LTDC_SYNC

- #define: ***LTDC_HORIZONTALSYNC (LTDC_SSCR_HSW >> 16)***

Horizontal synchronization width.

- #define: ***LTDC_VERTICALSYNC LTDC_SSCR_VSH***

Vertical synchronization heigh.

LTDC_VS_POLARITY

- #define: ***LTDC_VSPOLARITY_AL ((uint32_t)0x00000000)***

Vertical Synchronization is active low.

- #define: ***LTDC_VSPOLARITY_AH LTDC_GCR_VSPOL***

Vertical Synchronization is active high.

30 HAL NAND Generic Driver

30.1 NAND Firmware driver registers structures

30.1.1 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_InfoTypeDef Info***

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_InfoTypeDef NAND_HandleTypeDef::Info***
  - NAND characteristic information structure

30.1.2 NAND_AddressTypedef

NAND_AddressTypedef is defined in the stm32f4xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Zone***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypedef::Page***
  - NAND memory Page address
- ***uint16_t NAND_AddressTypedef::Zone***
  - NAND memory Zone address
- ***uint16_t NAND_AddressTypedef::Block***

– NAND memory Block address

30.1.3 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***

30.2 NAND Firmware driver API description

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

30.2.1 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()/HAL_NAND_Read_SpareArea(), HAL_NAND_Write_Page()/HAL_NAND_Write_SpareArea() 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()

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.

30.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

- ***HAL_NAND_Init()***
- ***HAL_NAND_DeInit()***
- ***HAL_NAND_MspInit()***
- ***HAL_NAND_MspDeInit()***
- ***HAL_NAND_IRQHandler()***
- ***HAL_NAND_ITCallback()***

30.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

- ***HAL_NAND_Read_ID()***
- ***HAL_NAND_Reset()***
- ***HAL_NAND_Read_Page()***
- ***HAL_NAND_Write_Page()***
- ***HAL_NAND_Read_SpareArea()***
- ***HAL_NAND_Write_SpareArea()***
- ***HAL_NAND_Erase_Block()***
- ***HAL_NAND_Read_Status()***
- ***HAL_NAND_Address_Inc()***

30.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

- ***HAL_NAND_ECC_Enable()***
- ***HAL_NAND_ECC_Disable()***
- ***HAL_NAND_GetECC()***

30.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

- ***HAL_NAND_GetState()***

30.2.6 Initialization and de-initialization functions

30.2.6.1 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**
Notes	• None.

30.2.6.2 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**
Notes	• None.

30.2.6.3 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.
Notes	• None.

30.2.6.4 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.
Notes	• None.

30.2.6.5 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**
Notes	• None.

30.2.6.6 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.
Notes	• None.

30.2.7 Input and Output functions

30.2.7.1 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**
Notes	• None.

30.2.7.2 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**
Notes	• None.

30.2.7.3 HAL_NAND_Read_Page

Function Name	**HAL_StatusTypeDef HAL_NAND_Read_Page (*NAND_HandleTypeDef ** hnand, *NAND_AddressTypedef ** pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)**
Function Description	Read Page(s) from NAND memory block.
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**
Notes	• None.

30.2.7.4 HAL_NAND_Write_Page

Function Name	**HAL_StatusTypeDef HAL_NAND_Write_Page (*NAND_HandleTypeDef ** hnand, *NAND_AddressTypedef ** pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)**
Function Description	Write Page(s) to NAND memory block.
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**
Notes	• None.

30.2.7.5 HAL_NAND_Read_SpareArea

Function Name	**HAL_StatusTypeDef HAL_NAND_Read_SpareArea (*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**
Notes	• None.

30.2.7.6 HAL_NAND_Write_SpareArea

Function Name	**HAL_StatusTypeDef HAL_NAND_Write_SpareArea (*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**
Notes	• None.

30.2.7.7 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**

Notes	• None.

30.2.7.8 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**
Notes	• None.

30.2.7.9 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 adress 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***
Notes	• None.

30.2.8 Control functions

30.2.8.1 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**
Notes	• None.

30.2.8.2 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**
Notes	• None.

30.2.8.3 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**
Notes	• None.

30.2.9 State functions

30.2.9.1 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**
Notes	• None.

30.3 NAND Firmware driver defines

30.3.1 NAND

NAND

NAND_Exported_Constants

- #define: ***NAND_DEVICE1 ((uint32_t)0x70000000)***
- #define: ***NAND_DEVICE2 ((uint32_t)0x80000000)***
- #define: ***NAND_WRITE_TIMEOUT ((uint32_t)0x01000000)***
- #define: ***CMD_AREA ((uint32_t)(1<<16))***

- #define: ***ADDR_AREA ((uint32_t)(1<<17))***
- #define: ***NAND_CMD_AREA_A ((uint8_t)0x00)***
- #define: ***NAND_CMD_AREA_B ((uint8_t)0x01)***
- #define: ***NAND_CMD_AREA_C ((uint8_t)0x50)***
- #define: ***NAND_VALID_ADDRESS ((uint32_t)0x00000100)***
- #define: ***NAND_INVALID_ADDRESS ((uint32_t)0x00000200)***
- #define: ***NAND_TIMEOUT_ERROR ((uint32_t)0x00000400)***
- #define: ***NAND_BUSY ((uint32_t)0x00000000)***
- #define: ***NAND_ERROR ((uint32_t)0x00000001)***
- #define: ***NAND_READY ((uint32_t)0x00000040)***

31 HAL NOR Generic Driver

31.1 NOR Firmware driver registers structures

31.1.1 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

31.1.2 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***

31.1.3 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 devices' 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

31.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

31.2.1 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()

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

31.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

- ***HAL_NOR_Init()***
- ***HAL_NOR_DeInit()***
- ***HAL_NOR_MspInit()***
- ***HAL_NOR_MspDeInit()***
- ***HAL_NOR_MspWait()***

31.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

- ***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()***

31.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

- ***HAL_NOR_WriteOperation_Enable()***
- ***HAL_NOR_WriteOperation_Disable()***

31.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

- *HAL_NOR_GetState()*
- *HAL_NOR_GetStatus()*

31.2.6 Initialization and de-initialization functions

31.2.6.1 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**
Notes	• None.

31.2.6.2 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**
Notes	• None.

31.2.6.3 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.
Notes	• None.

31.2.6.4 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.
Notes	• None.

31.2.6.5 HAL_NOR_MspWait

Function Name	**void HAL_NOR_MspWait (** ***NOR_HandleTypeDef **** **hnor, uint32_t Timeout)**
Function Description	NOR BSP Wait fro 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.
Notes	• None.

31.2.7 Input and Output functions

31.2.7.1 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**
Notes	• None.

31.2.7.2 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**
Notes	• None.

31.2.7.3 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**
Notes	• None.

31.2.7.4 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**
Notes	• None.

31.2.7.5 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 block of data from the FMC NOR memory.
Parameters	• **hnor :** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **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**
Notes	• None.

31.2.7.6 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 FMC NOR memory.
Parameters	• **hnor :** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **uwAddress :** NOR memory internal address from which the data • **pData :** pointer to source data buffer. • **uwBufferSize :** number of Half words to write. The maximum allowed
Return values	• **HAL status**
Notes	• None.

31.2.7.7 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**
Notes	• None.

31.2.7.8 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**

Notes	• None.

31.2.7.9 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**
Notes	• None.

31.2.8 Control functions

31.2.8.1 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**
Notes	• None.

31.2.8.2 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**
Notes	• None.

31.2.9 State functions

31.2.9.1 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**
Notes	• None.

31.2.9.2 HAL_NOR_GetStatus

Function Name	**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 progamming Timeout
Return values	• **NOR_Status : The returned value can be: NOR_SUCCESS, NOR_ERROR or NOR_TIMEOUT**
Notes	• None.

31.3 NOR Firmware driver defines

31.3.1 NOR

NOR

NOR_Exported_Constants

- #define: ***MC_ADDRESS ((uint16_t)0x0000)***
- #define: ***DEVICE_CODE1_ADDR ((uint16_t)0x0001)***
- #define: ***DEVICE_CODE2_ADDR ((uint16_t)0x000E)***
- #define: ***DEVICE_CODE3_ADDR ((uint16_t)0x000F)***
- #define: ***CFI1_ADDRESS ((uint16_t)0x61)***
- #define: ***CFI2_ADDRESS ((uint16_t)0x62)***
- #define: ***CFI3_ADDRESS ((uint16_t)0x63)***
- #define: ***CFI4_ADDRESS ((uint16_t)0x64)***
- #define: ***NOR_TMEOUT ((uint16_t)0xFFFF)***
- #define: ***NOR_MEMORY_8B***

- #define: ***NOR_MEMORY_ADRESS ((uint32_t)0x60000000)***

32 HAL PCCARD Generic Driver

32.1 PCCARD Firmware driver registers structures

32.1.1 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

32.2 PCCARD Firmware driver API description

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

32.2.1 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() with control and timing parameters for both common and attribute spaces.
- Read PCCARD/compact flash memory maker and device IDs using the function 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_CF_Read_Sector()/HAL_CF_Write_Sector(), to read/write sector.
- Perform PCCARD/compact flash Reset chip operation using the function HAL_CF_Reset().
- Perform PCCARD/compact flash erase sector operation using the function HAL_CF_Erase_Sector().

- Read the PCCARD/compact flash status operation using the function HAL_CF_ReadStatus().
- You can monitor the PCCARD/compact flash device HAL state by calling the function HAL_PCCARD_GetState()

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.

32.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

- ***HAL_PCCARD_Init()***
- ***HAL_PCCARD_DeInit()***
- ***HAL_PCCARD_MspInit()***
- ***HAL_PCCARD_MspDeInit()***

32.2.3 PCCARD Input and Output functions

This section provides functions allowing to use and control the PCCARD memory

- ***HAL_CF_Read_ID()***
- ***HAL_CF_Read_Sector()***
- ***HAL_CF_Write_Sector()***
- ***HAL_CF_Erase_Sector()***
- ***HAL_CF_Reset()***
- ***HAL_PCCARD_IRQHandler()***
- ***HAL_PCCARD_ITCallback()***

32.2.4 PCCARD State functions

This subsection permits to get in run-time the status of the PCCARD controller and the data flow.

- ***HAL_PCCARD_GetState()***
- ***HAL_CF_GetStatus()***
- ***HAL_CF_ReadStatus()***

32.2.5 Initialization and de-initialization functions

32.2.5.1 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**
Notes	• None.

32.2.5.2 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**
Notes	• None.

32.2.5.3 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.
Notes	• None.

32.2.5.4 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.
Notes	• None.

32.2.6 Input and Output functions

32.2.6.1 HAL_CF_Read_ID

Function Name	**HAL_StatusTypeDef HAL_CF_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**
Notes	• None.

32.2.6.2 HAL_CF_Read_Sector

Function Name	**HAL_StatusTypeDef HAL_CF_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 CF status
Return values	• **HAL status**
Notes	• None.

32.2.6.3 HAL_CF_Write_Sector

Function Name	**HAL_StatusTypeDef HAL_CF_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 CF status
Return values	• **HAL status**
Notes	• None.

32.2.6.4 HAL_CF_Erase_Sector

Function Name	**HAL_StatusTypeDef HAL_CF_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 CF status
Return values	• **HAL status**
Notes	• None.

32.2.6.5 HAL_CF_Reset

Function Name	**HAL_StatusTypeDef HAL_CF_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**
Notes	• None.

32.2.6.6 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**
Notes	• None.

32.2.6.7 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.
Notes	• None.

32.2.7 State functions

32.2.7.1 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**
Notes	• None.

32.2.7.2 HAL_CF_GetStatus

Function Name	**CF_StatusTypedef HAL_CF_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 CF 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
Notes	• None.

32.2.7.3 HAL_CF_ReadStatus

Function Name	**CF_StatusTypedef HAL_CF_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 gererates error***
Notes	• None.

32.3 PCCARD Firmware driver defines

32.3.1 PCCARD

PCCARD

PCCARD_Exported_Constants

- #define: ***CF_DEVICE_ADDRESS ((uint32_t)0x90000000)***

- #define: ***CF_ATTRIBUTE_SPACE_ADDRESS ((uint32_t)0x98000000)***

- #define: ***CF_COMMON_SPACE_ADDRESS CF_DEVICE_ADDRESS***

- #define: ***CF_IO_SPACE_ADDRESS ((uint32_t)0x9C000000)***
- #define: ***CF_IO_SPACE_PRIMARY_ADDR ((uint32_t)0x9C0001F0)***
- #define: ***CF_DATA ((uint8_t)0x00)***
- #define: ***CF_SECTOR_COUNT ((uint8_t)0x02)***
- #define: ***CF_SECTOR_NUMBER ((uint8_t)0x03)***
- #define: ***CF_CYLINDER_LOW ((uint8_t)0x04)***
- #define: ***CF_CYLINDER_HIGH ((uint8_t)0x05)***
- #define: ***CF_CARD_HEAD ((uint8_t)0x06)***
- #define: ***CF_STATUS_CMD ((uint8_t)0x07)***
- #define: ***CF_STATUS_CMD_ALTERNATE ((uint8_t)0x0E)***
- #define: ***CF_COMMON_DATA_AREA ((uint16_t)0x0400)***
- #define: ***CF_READ_SECTOR_CMD ((uint8_t)0x20)***

- #define: ***CF_WRITE_SECTOR_CMD ((uint8_t)0x30)***
- #define: ***CF_ERASE_SECTOR_CMD ((uint8_t)0xC0)***
- #define: ***CF_IDENTIFY_CMD ((uint8_t)0xEC)***
- #define: ***CF_TIMEOUT_ERROR ((uint8_t)0x60)***
- #define: ***CF_BUSY ((uint8_t)0x80)***
- #define: ***CF_PROGR ((uint8_t)0x01)***
- #define: ***CF_READY ((uint8_t)0x40)***
- #define: ***CF_SECTOR_SIZE ((uint32_t)255)***

33 HAL PCD Generic Driver

33.1 PCD Firmware driver registers structures

33.1.1 PCD_HandleTypeDef

PCD_HandleTypeDef is defined in the stm32f4xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***uint32_t Setup***
- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
  – Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
  – PCD required parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[15]***
  – IN endpoint parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::OUT_ep[15]***
  – OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
  – PCD peripheral status
- ***__IO PCD_StateTypeDef PCD_HandleTypeDef::State***
  – PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
  – Setup packet buffer
- ***void* PCD_HandleTypeDef::pData***
  – Pointer to upper stack Handler

33.2 PCD Firmware driver API description

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

33.2.1 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 HCD 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
      – __OTGFS-OTG_CLK_ENABLE()/__OTGHS-OTG_CLK_ENABLE();
      – __OTGHSULPI_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 HCD transmission and reception:
   a. HAL_PCD_Start();

33.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- ***HAL_PCD_Init()***
- ***HAL_PCD_DeInit()***
- ***HAL_PCD_MspInit()***
- ***HAL_PCD_MspDeInit()***

33.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

- ***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()***

33.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

- ***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_SetTxFiFo()*
- *HAL_PCD_SetRxFiFo()*
- *HAL_PCD_ActiveRemoteWakeup()*
- *HAL_PCD_DeActiveRemoteWakeup()*

33.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_PCD_GetState()*

33.2.6 Initialization and de-initialization functions

33.2.6.1 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 create the associated handle.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.6.2 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**
Notes	• None.

33.2.6.3 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.
Notes	• None.

33.2.6.4 HAL_PCD_MspDeInit

Function Name	**void HAL_PCD_MspDeInit (** ***PCD_HandleTypeDef **** **hpcd)**
Function Description	DeInitializes PCD MSP.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7 IO operation functions

33.2.7.1 HAL_PCD_Start

Function Name	**HAL_StatusTypeDef HAL_PCD_Start (** ***PCD_HandleTypeDef **** **hpcd)**
Function Description	Start The USB OTG Device.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.7.2 HAL_PCD_Stop

Function Name	**HAL_StatusTypeDef HAL_PCD_Stop (*PCD_HandleTypeDef ** hpcd)**
Function Description	Stop The USB OTG Device.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.7.3 HAL_PCD_IRQHandler

Function Name	**void HAL_PCD_IRQHandler (*PCD_HandleTypeDef ** hpcd)**
Function Description	This function handles PCD interrupt request.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.7.4 HAL_PCD_DataOutStageCallback

Function Name	**void HAL_PCD_DataOutStageCallback (*PCD_HandleTypeDef ** hpcd, uint8_t epnum)**
Function Description	Data out stage callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.5 HAL_PCD_DataInStageCallback

Function Name	**void HAL_PCD_DataInStageCallback (*PCD_HandleTypeDef ** hpcd, uint8_t epnum)**
Function Description	Data IN stage callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.6 HAL_PCD_SetupStageCallback

Function Name	**void HAL_PCD_SetupStageCallback (*PCD_HandleTypeDef ** hpcd)**
Function Description	Setup stage callback.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.7 HAL_PCD_SOFCallback

Function Name	**void HAL_PCD_SOFCallback (*PCD_HandleTypeDef ** hpcd)**
Function Description	USB Start Of Frame callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.8 HAL_PCD_ResetCallback

Function Name	**void HAL_PCD_ResetCallback (*PCD_HandleTypeDef ** hpcd)**
Function Description	USB Reset callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.9 HAL_PCD_SuspendCallback

Function Name	**void HAL_PCD_SuspendCallback (*PCD_HandleTypeDef ** hpcd)**
Function Description	Suspend event callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.10 HAL_PCD_ResumeCallback

Function Name	**void HAL_PCD_ResumeCallback (*PCD_HandleTypeDef ** hpcd)**
Function Description	Resume event callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.11 HAL_PCD_ISOOUTIncompleteCallback

Function Name	**void HAL_PCD_ISOOUTIncompleteCallback (*PCD_HandleTypeDef ** hpcd, uint8_t epnum)**
Function Description	Incomplete ISO OUT callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.12 HAL_PCD_ISOINIncompleteCallback

Function Name	**void HAL_PCD_ISOINIncompleteCallback (*PCD_HandleTypeDef ** hpcd, uint8_t epnum)**
Function Description	Incomplete ISO IN callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.13 HAL_PCD_ConnectCallback

Function Name	**void HAL_PCD_ConnectCallback (*PCD_HandleTypeDef ** hpcd)**
Function Description	Connection event callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.7.14 HAL_PCD_DisconnectCallback

Function Name	**void HAL_PCD_DisconnectCallback (** ***PCD_HandleTypeDef **** **hpcd)**
Function Description	Disconnection event callbacks.
Parameters	• **hpcd :** PCD handle
Return values	• None.
Notes	• None.

33.2.8 Peripheral Control functions

33.2.8.1 HAL_PCD_DevConnect

Function Name	**HAL_StatusTypeDef HAL_PCD_DevConnect (** ***PCD_HandleTypeDef **** **hpcd)**
Function Description	Send an amount of data in blocking mode.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.8.2 HAL_PCD_DevDisconnect

Function Name	**HAL_StatusTypeDef HAL_PCD_DevDisconnect (** ***PCD_HandleTypeDef **** **hpcd)**
Function Description	Send an amount of data in blocking mode.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.8.3 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**
Notes	• None.

33.2.8.4 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 packert size • **ep_type :** endpoint type
Return values	• **HAL status**
Notes	• None.

33.2.8.5 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**
Notes	• None.

33.2.8.6 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**
Notes	• None.

33.2.8.7 HAL_PCD_EP_GetRxCount

Function Name	**uint16_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**
Notes	• None.

33.2.8.8 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**
Notes	• None.

33.2.8.9 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**
Notes	• None.

33.2.8.10 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**
Notes	• None.

33.2.8.11 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**
Notes	• None.

33.2.8.12 HAL_PCD_SetTxFiFo

Function Name	**HAL_StatusTypeDef HAL_PCD_SetTxFiFo (*PCD_HandleTypeDef ** hpcd, uint8_t fifo, uint16_t size)**
Function Description	Update FIFO configuration.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.8.13 HAL_PCD_SetRxFiFo

Function Name	**HAL_StatusTypeDef HAL_PCD_SetRxFiFo (*PCD_HandleTypeDef ** hpcd, uint16_t size)**
Function Description	Update FIFO configuration.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.8.14 HAL_PCD_ActiveRemoteWakeup

Function Name	**HAL_StatusTypeDef HAL_PCD_ActiveRemoteWakeup (*PCD_HandleTypeDef* * hpcd)**
Function Description	HAL_PCD_ActiveRemoteWakeup : active remote wakeup signalling.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.8.15 HAL_PCD_DeActiveRemoteWakeup

Function Name	**HAL_StatusTypeDef HAL_PCD_DeActiveRemoteWakeup (*PCD_HandleTypeDef* * hpcd)**
Function Description	HAL_PCD_DeActiveRemoteWakeup : de-active remote wakeup signalling.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL status**
Notes	• None.

33.2.9 Peripheral State functions

33.2.9.1 HAL_PCD_GetState

Function Name	**PCD_StateTypeDef HAL_PCD_GetState (*PCD_HandleTypeDef* * hpcd)**
Function Description	Return the PCD state.
Parameters	• **hpcd :** PCD handle
Return values	• **HAL state**
Notes	• None.

33.3 PCD Firmware driver defines

33.3.1 PCD

PCD

PCD_Interrupt_Clock

- #define: ***USB_FS_EXTI_TRIGGER_RISING_EDGE ((uint32_t)0x08)***
- #define: ***USB_FS_EXTI_TRIGGER_FALLING_EDGE ((uint32_t)0x0C)***
- #define: ***USB_FS_EXTI_TRIGGER_BOTH_EDGE ((uint32_t)0x10)***
- #define: ***USB_HS_EXTI_TRIGGER_RISING_EDGE ((uint32_t)0x08)***
- #define: ***USB_HS_EXTI_TRIGGER_FALLING_EDGE ((uint32_t)0x0C)***
- #define: ***USB_HS_EXTI_TRIGGER_BOTH_EDGE ((uint32_t)0x10)***
- #define: ***USB_HS_EXTI_LINE_WAKEUP ((uint32_t)0x00100000)***

External interrupt line 20 Connected to the USB HS EXTI Line

- #define: ***USB_FS_EXTI_LINE_WAKEUP ((uint32_t)0x00040000)***

External interrupt line 18 Connected to the USB FS EXTI Line

- #define: ***__HAL_USB_HS_EXTI_ENABLE_IT EXTI->IMR |= (USB_HS_EXTI_LINE_WAKEUP)***
- #define: ***__HAL_USB_HS_EXTI_DISABLE_IT EXTI->IMR &= ~(USB_HS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_HS_EXTI_GET_FLAG EXTI->PR & (USB_HS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_HS_EXTI_CLEAR_FLAG EXTI->PR = (USB_HS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_HS_EXTI_SET_RISING_EGDE_TRIGGER EXTI->FTSR &= ~(USB_HS_EXTI_LINE_WAKEUP);\ EXTI->RTSR |= USB_HS_EXTI_LINE_WAKEUP***

- #define: ***__HAL_USB_HS_EXTI_SET_FALLING_EGDE_TRIGGER EXTI->FTSR |= (USB_HS_EXTI_LINE_WAKEUP);\ EXTI->RTSR &= ~(USB_HS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_HS_EXTI_SET_FALLINGRISING_TRIGGER EXTI->RTSR &= ~(USB_HS_EXTI_LINE_WAKEUP);\ EXTI->FTSR &= ~(USB_HS_EXTI_LINE_WAKEUP;)\ EXTI->RTSR |= USB_HS_EXTI_LINE_WAKEUP;\ EXTI->FTSR |= USB_HS_EXTI_LINE_WAKEUP***

- #define: ***__HAL_USB_FS_EXTI_ENABLE_IT EXTI->IMR |= USB_FS_EXTI_LINE_WAKEUP***

- #define: ***__HAL_USB_FS_EXTI_DISABLE_IT EXTI->IMR &= ~(USB_FS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_FS_EXTI_GET_FLAG EXTI->PR & (USB_FS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_FS_EXTI_CLEAR_FLAG EXTI->PR = USB_FS_EXTI_LINE_WAKEUP***

- #define: ***__HAL_USB_FS_EXTI_SET_RISING_EGDE_TRIGGER EXTI->FTSR &= ~(USB_FS_EXTI_LINE_WAKEUP);\ EXTI->RTSR |= USB_FS_EXTI_LINE_WAKEUP***

- #define: ***__HAL_USB_FS_EXTI_SET_FALLING_EGDE_TRIGGER EXTI->FTSR |= (USB_FS_EXTI_LINE_WAKEUP);\ EXTI->RTSR &= ~(USB_FS_EXTI_LINE_WAKEUP)***

- #define: ***__HAL_USB_FS_EXTI_SET_FALLINGRISING_TRIGGER EXTI->RTSR &= ~(USB_FS_EXTI_LINE_WAKEUP);\ EXTI->FTSR &= ~(USB_FS_EXTI_LINE_WAKEUP);\ EXTI->RTSR |= USB_FS_EXTI_LINE_WAKEUP;\ EXTI->FTSR |= USB_FS_EXTI_LINE_WAKEUP***

PCD_PHY_Module

- #define: ***PCD_PHY_ULPI 1***

- #define: ***PCD_PHY_EMBEDDED 2***

PCD_Speed

- #define: ***PCD_SPEED_HIGH 0***

- #define: ***PCD_SPEED_HIGH_IN_FULL 1***

- #define: ***PCD_SPEED_FULL 2***

34 HAL PWR Generic Driver

34.1 PWR Firmware driver registers structures

34.1.1 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***

34.1.2 PWR_TypeDef

PWR_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t CSR***

Field Documentation

- ***__IO uint32_t PWR_TypeDef::CR***
  - PWR power control register, Address offset: 0x00
- ***__IO uint32_t PWR_TypeDef::CSR***
  - PWR power control/status register, Address offset: 0x04

34.2 PWR Firmware driver API description

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

34.2.1 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 __PWR_CLK_ENABLE() macro.
- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.
- ***HAL_PWR_DeInit()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

34.2.2 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_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp 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 only one WakeUp pin: WakeUp Pin 1 on PA.00.

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 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_PWR_EnableFlashPowerDown() function. It can be switched on again by software after exiting the Stop mode using the HAL_PWR_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 wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the Stop 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 WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- ***HAL_PWR_PVDConfig()***
- ***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()***

34.2.3 Initialization and de-initialization functions

34.2.3.1 HAL_PWR_DeInit

Function Name	**void HAL_PWR_DeInit (void)**
Function Description	Deinitializes the HAL PWR peripheral registers to their default reset values.
Parameters	• None.
Return values	• None.
Notes	• None.

34.2.3.2 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).
Parameters	• None.
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.

34.2.3.3 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).
Parameters	• None.
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.

34.2.4 Peripheral Control functions

34.2.4.1 HAL_PWR_PVDConfig

Function Name	**void HAL_PWR_PVDConfig (*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.

34.2.4.2 HAL_PWR_EnablePVD

Function Name	**void HAL_PWR_EnablePVD (void)**
Function Description	Enables the Power Voltage Detector(PVD).
Parameters	• None.
Return values	• None.
Notes	• None.

34.2.4.3 HAL_PWR_DisablePVD

Function Name	**void HAL_PWR_DisablePVD (void)**

Function Description	Disables the Power Voltage Detector(PVD).
Parameters	• None.
Return values	• None.
Notes	• None.

34.2.4.4 HAL_PWR_EnableWakeUpPin

Function Name	**void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)**
Function Description	Enables the WakeUp PINx functionality.
Parameters	• **WakeUpPinx :** Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: – ***PWR_WAKEUP_PIN1 :***
Return values	• None.
Notes	• None.

34.2.4.5 HAL_PWR_DisableWakeUpPin

Function Name	**void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)**
Function Description	Disables the WakeUp PINx functionality.
Parameters	• **WakeUpPinx :** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: – ***PWR_WAKEUP_PIN1 :***
Return values	• None.
Notes	• None.

34.2.4.6 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
Parameters	• **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.

34.2.4.7 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 wakeup 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.

34.2.4.8 HAL_PWR_EnterSTANDBYMode

Function Name	**void HAL_PWR_EnterSTANDBYMode (void)**
Function Description	Enters Standby mode.
Parameters	• None.
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.

34.2.4.9 HAL_PWR_PVD_IRQHandler

Function Name	**void HAL_PWR_PVD_IRQHandler (void)**
Function Description	This function handles the PWR PVD interrupt request.
Parameters	• None.
Return values	• None.
Notes	• This API should be called under the PVD_IRQHandler().

34.2.4.10 HAL_PWR_PVDCallback

Function Name	**void HAL_PWR_PVDCallback (void)**
Function Description	PWR PVD interrupt callback.
Parameters	• None.
Return values	• None.
Notes	• None.

34.3 PWR Firmware driver defines

34.3.1 PWR

PWR

PWR_Flag

- #define: ***PWR_FLAG_WU PWR_CSR_WUF***
- #define: ***PWR_FLAG_SB PWR_CSR_SBF***
- #define: ***PWR_FLAG_PVDO PWR_CSR_PVDO***
- #define: ***PWR_FLAG_BRR PWR_CSR_BRR***
- #define: ***PWR_FLAG_VOSRDY PWR_CSR_VOSRDY***

PWR_PVD_detection_level

- #define: ***PWR_PVDLEVEL_0 PWR_CR_PLS_LEV0***
- #define: ***PWR_PVDLEVEL_1 PWR_CR_PLS_LEV1***

- #define: ***PWR_PVDLEVEL_2 PWR_CR_PLS_LEV2***

- #define: ***PWR_PVDLEVEL_3 PWR_CR_PLS_LEV3***

- #define: ***PWR_PVDLEVEL_4 PWR_CR_PLS_LEV4***

- #define: ***PWR_PVDLEVEL_5 PWR_CR_PLS_LEV5***

- #define: ***PWR_PVDLEVEL_6 PWR_CR_PLS_LEV6***

- #define: ***PWR_PVDLEVEL_7 PWR_CR_PLS_LEV7***

PWR_PVD_Mode

- #define: ***PWR_MODE_EVT ((uint32_t)0x00000000)***

No Interrupt

- #define: ***PWR_MODE_IT_RISING ((uint32_t)0x00000001)***

External Interrupt Mode with Rising edge trigger detection

- #define: ***PWR_MODE_IT_FALLING ((uint32_t)0x00000002)***

External Interrupt Mode with Falling edge trigger detection

- #define: ***PWR_MODE_IT_RISING_FALLING ((uint32_t)0x00000003)***

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_Regulator_state_in_STOP_mode

- #define: ***PWR_MAINREGULATOR_ON ((uint32_t)0x00000000)***

- #define: ***PWR_LOWPOWERREGULATOR_ON PWR_CR_LPDS***

PWR_Regulator_Voltage_Scale

- #define: ***PWR_REGULATOR_VOLTAGE_SCALE1 ((uint32_t)0x0000C000)***
- #define: ***PWR_REGULATOR_VOLTAGE_SCALE2 ((uint32_t)0x00008000)***
- #define: ***PWR_REGULATOR_VOLTAGE_SCALE3 ((uint32_t)0x00004000)***

PWR_SLEEP_mode_entry

- #define: ***PWR_SLEEPENTRY_WFI ((uint8_t)0x01)***
- #define: ***PWR_SLEEPENTRY_WFE ((uint8_t)0x02)***

PWR_STOP_mode_entry

- #define: ***PWR_STOPENTRY_WFI ((uint8_t)0x01)***
- #define: ***PWR_STOPENTRY_WFE ((uint8_t)0x02)***

PWR_WakeUp_Pins

- #define: ***PWR_WAKEUP_PIN1 PWR_CSR_EWUP***

35 HAL PWR Extension Driver

35.1 PWREx Firmware driver API description

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

35.1.1 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_PWR_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. 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_PWR_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 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 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 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.
- ***HAL_PWREx_EnableBkUpReg()***
- ***HAL_PWREx_DisableBkUpReg()***
- ***HAL_PWREx_EnableFlashPowerDown()***
- ***HAL_PWREx_DisableFlashPowerDown()***
- ***HAL_PWREx_ActivateOverDrive()***
- ***HAL_PWREx_DeactivateOverDrive()***

35.1.2 Peripheral Extended features functions

35.1.2.1 HAL_PWREx_EnableBkUpReg

Function Name	**HAL_StatusTypeDef HAL_PWREx_EnableBkUpReg (void)**
Function Description	Enables the Backup Regulator.
Parameters	• None.
Return values	• **HAL status**
Notes	• None.

35.1.2.2 HAL_PWREx_DisableBkUpReg

Function Name	**HAL_StatusTypeDef HAL_PWREx_DisableBkUpReg (void)**
Function Description	Disables the Backup Regulator.
Parameters	• None.
Return values	• **HAL status**
Notes	• None.

35.1.2.3 HAL_PWREx_EnableFlashPowerDown

Function Name	**void HAL_PWREx_EnableFlashPowerDown (void)**
Function Description	Enables the Flash Power Down in Stop mode.
Parameters	• None.
Return values	• None.
Notes	• None.

35.1.2.4 HAL_PWREx_DisableFlashPowerDown

Function Name	**void HAL_PWREx_DisableFlashPowerDown (void)**
Function Description	Disables the Flash Power Down in Stop mode.
Parameters	• None.
Return values	• None.
Notes	• None.

35.1.2.5 HAL_PWREx_ActivateOverDrive

Function Name	**HAL_StatusTypeDef HAL_PWREx_ActivateOverDrive (void)**
Function Description	Activates the Over-Drive mode.
Parameters	• None.
Return values	• **HAL status**
Notes	• These macros can be used only for STM32F42xx/STM32F43xx 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.

35.1.2.6 HAL_PWREx_DeactivateOverDrive

Function Name	**HAL_StatusTypeDef HAL_PWREx_DeactivateOverDrive (void)**
Function Description	Deactivates the Over-Drive mode.
Parameters	• None.
Return values	• **HAL status**
Notes	• These macros can be used only for STM32F42xx/STM32F43xx 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.

35.2 PWREx Firmware driver defines

35.2.1 PWREx

PWREx

PWREx_Over_Under_Drive_Flag

- #define: ***PWR_FLAG_ODRDY PWR_CSR_ODRDY***
- #define: ***PWR_FLAG_ODSWRDY PWR_CSR_ODSWRDY***
- #define: ***PWR_FLAG_UDRDY PWR_CSR_UDSWRDY***

36 HAL RCC Generic Driver

36.1 RCC Firmware driver registers structures

36.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32f4xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t PLLQ***

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 = 192 and Max_Data = 432
- ***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 = 0 and Max_Data = 63

36.1.2 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***

36.1.3 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 calibration trimming value. 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

36.2 RCC Firmware driver API description

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

36.2.1 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)

36.2.2 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 clockis 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. All the peripheral clocks are derived from the System clock (SYSCLK) except: I2S: the I2S clock can be derived either from a specific PLL (PLLI2S) or from an external clock mapped on the I2S_CKIN pin. You have to use __HAL_RCC_PLLI2S_CONFIG() macro to configure this clock. SAI: the SAI clock can be derived either from a specific PLL (PLLI2S) or (PLLSAI) or from an external clock mapped on the I2S_CKIN pin. You have to use __HAL_RCC_PLLI2S_CONFIG() macro to configure this clock. RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use __HAL_RCC_RTC_CONFIG() and __HAL_RCC_RTC_ENABLE() macros to configure this clock. USB OTG FS, SDIO and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly, while the SDIO require a frequency equal or lower than to 48. This clock is derived of the main PLL through PLLQ divider. IWDG clock which is always the LSI clock.
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 and STM32F43xxx 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).

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

36.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

- ***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_CCSCallback()*

36.2.4 Initialization and de-initialization functions

36.2.4.1 HAL_RCC_DeInit

Function Name	**void HAL_RCC_DeInit (void)**
Function Description	Resets the RCC clock configuration to the default reset state.
Parameters	• None.
Return values	• None.
Notes	• The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL and PLLI2S 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

36.2.4.2 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.

36.2.4.3 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")

36.2.5 Peripheral Control functions

36.2.5.1 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
- *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.

36.2.5.2 HAL_RCC_EnableCSS

Function Name	**void HAL_RCC_EnableCSS (void)**
Function Description	Enables the Clock Security System.
Parameters	• None.
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.

36.2.5.3 HAL_RCC_DisableCSS

Function Name	**void HAL_RCC_DisableCSS (void)**
Function Description	Disables the Clock Security System.
Parameters	• None.
Return values	• None.
Notes	• None.

36.2.5.4 HAL_RCC_GetSysClockFreq

Function Name	**uint32_t HAL_RCC_GetSysClockFreq (void)**
Function Description	Returns the SYSCLK frequency.
Parameters	• None.
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.

36.2.5.5 HAL_RCC_GetHCLKFreq

Function Name	**uint32_t HAL_RCC_GetHCLKFreq (void)**
Function Description	Returns the HCLK frequency.
Parameters	• None.
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

36.2.5.6 HAL_RCC_GetPCLK1Freq

Function Name	**uint32_t HAL_RCC_GetPCLK1Freq (void)**
Function Description	Returns the PCLK1 frequency.
Parameters	• None.
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.

36.2.5.7 HAL_RCC_GetPCLK2Freq

Function Name	**uint32_t HAL_RCC_GetPCLK2Freq (void)**
Function Description	Returns the PCLK2 frequency.
Parameters	• None.
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.

36.2.5.8 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.
Notes	• None.

36.2.5.9 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_OscInitStruct :** pointer to an RCC_ClkInitTypeDef structure that will be configured. • **pFLatency :** Pointer on the Flash Latency.
Return values	• None.
Notes	• None.

36.2.5.10 HAL_RCC_NMI_IRQHandler

Function Name	**void HAL_RCC_NMI_IRQHandler (void)**
Function Description	This function handles the RCC CSS interrupt request.
Parameters	• None.
Return values	• None.
Notes	• This API should be called under the NMI_Handler().

36.2.5.11 HAL_RCC_CCSCallback

Function Name	**void HAL_RCC_CCSCallback (void)**
Function Description	RCC Clock Security System interrupt callback.
Parameters	• **none :**
Return values	• **none**
Notes	• None.

36.3 RCC Firmware driver defines

36.3.1 RCC

RCC

RCC_AHB_Clock_Source

- #define: ***RCC_SYSCLK_DIV1 RCC_CFGR_HPRE_DIV1***
- #define: ***RCC_SYSCLK_DIV2 RCC_CFGR_HPRE_DIV2***
- #define: ***RCC_SYSCLK_DIV4 RCC_CFGR_HPRE_DIV4***
- #define: ***RCC_SYSCLK_DIV8 RCC_CFGR_HPRE_DIV8***

- #define: ***RCC_SYSCLK_DIV16 RCC_CFGR_HPRE_DIV16***
- #define: ***RCC_SYSCLK_DIV64 RCC_CFGR_HPRE_DIV64***
- #define: ***RCC_SYSCLK_DIV128 RCC_CFGR_HPRE_DIV128***
- #define: ***RCC_SYSCLK_DIV256 RCC_CFGR_HPRE_DIV256***
- #define: ***RCC_SYSCLK_DIV512 RCC_CFGR_HPRE_DIV512***

RCC_APB1_APB2_Clock_Source

- #define: ***RCC_HCLK_DIV1 RCC_CFGR_PPRE1_DIV1***
- #define: ***RCC_HCLK_DIV2 RCC_CFGR_PPRE1_DIV2***
- #define: ***RCC_HCLK_DIV4 RCC_CFGR_PPRE1_DIV4***
- #define: ***RCC_HCLK_DIV8 RCC_CFGR_PPRE1_DIV8***
- #define: ***RCC_HCLK_DIV16 RCC_CFGR_PPRE1_DIV16***

RCC_BitAddress_AliasRegion

- #define: ***RCC_OFFSET (RCC_BASE - PERIPH_BASE)***
- #define: ***RCC_CR_OFFSET (RCC_OFFSET + 0x00)***

- #define: ***HSION_BitNumber 0x00***
- #define: ***CR_HSION_BB (PERIPH_BB_BASE + (RCC_CR_OFFSET * 32) + (HSION_BitNumber * 4))***
- #define: ***CSSON_BitNumber 0x13***
- #define: ***CR_CSSON_BB (PERIPH_BB_BASE + (RCC_CR_OFFSET * 32) + (CSSON_BitNumber * 4))***
- #define: ***PLLON_BitNumber 0x18***
- #define: ***CR_PLLON_BB (PERIPH_BB_BASE + (RCC_CR_OFFSET * 32) + (PLLON_BitNumber * 4))***
- #define: ***PLLI2SON_BitNumber 0x1A***
- #define: ***CR_PLLI2SON_BB (PERIPH_BB_BASE + (RCC_CR_OFFSET * 32) + (PLLI2SON_BitNumber * 4))***
- #define: ***RCC_CFGR_OFFSET (RCC_OFFSET + 0x08)***
- #define: ***I2SSRC_BitNumber 0x17***
- #define: ***CFGR_I2SSRC_BB (PERIPH_BB_BASE + (RCC_CFGR_OFFSET * 32) + (I2SSRC_BitNumber * 4))***

- #define: ***RCC_BDCR_OFFSET (RCC_OFFSET + 0x70)***

- #define: ***RTCEN_BitNumber 0x0F***

- #define: ***BDCR_RTCEN_BB (PERIPH_BB_BASE + (RCC_BDCR_OFFSET * 32) + (RTCEN_BitNumber * 4))***

- #define: ***BDRST_BitNumber 0x10***

- #define: ***BDCR_BDRST_BB (PERIPH_BB_BASE + (RCC_BDCR_OFFSET * 32) + (BDRST_BitNumber * 4))***

- #define: ***RCC_CSR_OFFSET (RCC_OFFSET + 0x74)***

- #define: ***LSION_BitNumber 0x00***

- #define: ***CSR_LSION_BB (PERIPH_BB_BASE + (RCC_CSR_OFFSET * 32) + (LSION_BitNumber * 4))***

- #define: ***CR_BYTE2_ADDRESS ((uint32_t)0x40023802)***

- #define: ***CIR_BYTE1_ADDRESS ((uint32_t)(RCC_BASE + 0x0C + 0x01))***

- #define: ***CIR_BYTE2_ADDRESS ((uint32_t)(RCC_BASE + 0x0C + 0x02))***

- #define: ***BDCR_BYTE0_ADDRESS (PERIPH_BASE + RCC_BDCR_OFFSET)***
- #define: ***DBP_TIMEOUT_VALUE ((uint32_t)100)***
- #define: ***LSE_TIMEOUT_VALUE ((uint32_t)5000)***

RCC_Flag

- #define: ***RCC_FLAG_HSIRDY ((uint8_t)0x21)***
- #define: ***RCC_FLAG_HSERDY ((uint8_t)0x31)***
- #define: ***RCC_FLAG_PLLRDY ((uint8_t)0x39)***
- #define: ***RCC_FLAG_PLLI2SRDY ((uint8_t)0x3B)***
- #define: ***RCC_FLAG_LSERDY ((uint8_t)0x41)***
- #define: ***RCC_FLAG_LSIRDY ((uint8_t)0x61)***
- #define: ***RCC_FLAG_BORRST ((uint8_t)0x79)***
- #define: ***RCC_FLAG_PINRST ((uint8_t)0x7A)***
- #define: ***RCC_FLAG_PORRST ((uint8_t)0x7B)***

- #define: ***RCC_FLAG_SFTRST ((uint8_t)0x7C)***
- #define: ***RCC_FLAG_IWDGRST ((uint8_t)0x7D)***
- #define: ***RCC_FLAG_WWDGRST ((uint8_t)0x7E)***
- #define: ***RCC_FLAG_LPWRRST ((uint8_t)0x7F)***

RCC_HSE_Config

- #define: ***RCC_HSE_OFF ((uint8_t)0x00)***
- #define: ***RCC_HSE_ON ((uint8_t)0x01)***
- #define: ***RCC_HSE_BYPASS ((uint8_t)0x05)***

RCC_HSI_Config

- #define: ***RCC_HSI_OFF ((uint8_t)0x00)***
- #define: ***RCC_HSI_ON ((uint8_t)0x01)***

RCC_I2S_Clock_Source

- #define: ***RCC_I2SCLKSOURCE_PLLI2S ((uint32_t)0x00000000)***
- #define: ***RCC_I2SCLKSOURCE_EXT ((uint32_t)0x00000001)***

RCC_Interrupt

- #define: ***RCC_IT_LSIRDY ((uint8_t)0x01)***
- #define: ***RCC_IT_LSERDY ((uint8_t)0x02)***
- #define: ***RCC_IT_HSIRDY ((uint8_t)0x04)***
- #define: ***RCC_IT_HSERDY ((uint8_t)0x08)***
- #define: ***RCC_IT_PLLRDY ((uint8_t)0x10)***
- #define: ***RCC_IT_PLLI2SRDY ((uint8_t)0x20)***
- #define: ***RCC_IT_CSS ((uint8_t)0x80)***

RCC_LSE_Config

- #define: ***RCC_LSE_OFF ((uint8_t)0x00)***
- #define: ***RCC_LSE_ON ((uint8_t)0x01)***
- #define: ***RCC_LSE_BYPASS ((uint8_t)0x05)***

RCC_LSI_Config

- #define: ***RCC_LSI_OFF ((uint8_t)0x00)***

- #define: ***RCC_LSI_ON ((uint8_t)0x01)***

RCC_MCO1_Clock_Source

- #define: ***RCC_MCO1SOURCE_HSI ((uint32_t)0x00000000)***
- #define: ***RCC_MCO1SOURCE_LSE RCC_CFGR_MCO1_0***
- #define: ***RCC_MCO1SOURCE_HSE RCC_CFGR_MCO1_1***
- #define: ***RCC_MCO1SOURCE_PLLCLK RCC_CFGR_MCO1***

RCC_MCO2_Clock_Source

- #define: ***RCC_MCO2SOURCE_SYSCLK ((uint32_t)0x00000000)***
- #define: ***RCC_MCO2SOURCE_PLLI2SCLK RCC_CFGR_MCO2_0***
- #define: ***RCC_MCO2SOURCE_HSE RCC_CFGR_MCO2_1***
- #define: ***RCC_MCO2SOURCE_PLLCLK RCC_CFGR_MCO2***

RCC_MCOx_Clock_Prescaler

- #define: ***RCC_MCODIV_1 ((uint32_t)0x00000000)***
- #define: ***RCC_MCODIV_2 RCC_CFGR_MCO1PRE_2***

- #define: ***RCC_MCODIV_3 ((uint32_t)RCC_CFGR_MCO1PRE_0 | RCC_CFGR_MCO1PRE_2)***
- #define: ***RCC_MCODIV_4 ((uint32_t)RCC_CFGR_MCO1PRE_1 | RCC_CFGR_MCO1PRE_2)***
- #define: ***RCC_MCODIV_5 RCC_CFGR_MCO1PRE***

RCC_MCO_Index

- #define: ***RCC_MCO1 ((uint32_t)0x00000000)***
- #define: ***RCC_MCO2 ((uint32_t)0x00000001)***

RCC_Oscillator_Type

- #define: ***RCC_OSCILLATORTYPE_NONE ((uint32_t)0x00000000)***
- #define: ***RCC_OSCILLATORTYPE_HSE ((uint32_t)0x00000001)***
- #define: ***RCC_OSCILLATORTYPE_HSI ((uint32_t)0x00000002)***
- #define: ***RCC_OSCILLATORTYPE_LSE ((uint32_t)0x00000004)***
- #define: ***RCC_OSCILLATORTYPE_LSI ((uint32_t)0x00000008)***

RCC_PLLP_Clock_Divider

- #define: ***RCC_PLLP_DIV2 ((uint32_t)0x00000002)***
- #define: ***RCC_PLLP_DIV4 ((uint32_t)0x00000004)***
- #define: ***RCC_PLLP_DIV6 ((uint32_t)0x00000006)***
- #define: ***RCC_PLLP_DIV8 ((uint32_t)0x00000008)***

RCC_PLL_Clock_Source

- #define: ***RCC_PLLSOURCE_HSI RCC_PLLCFGR_PLLSRC_HSI***
- #define: ***RCC_PLLSOURCE_HSE RCC_PLLCFGR_PLLSRC_HSE***

RCC_PLL_Config

- #define: ***RCC_PLL_NONE ((uint8_t)0x00)***
- #define: ***RCC_PLL_OFF ((uint8_t)0x01)***
- #define: ***RCC_PLL_ON ((uint8_t)0x02)***

RCC_RTC_Clock_Source

- #define: ***RCC_RTCCLKSOURCE_LSE ((uint32_t)0x00000100)***
- #define: ***RCC_RTCCLKSOURCE_LSI ((uint32_t)0x00000200)***

- #define: ***RCC_RTCCLKSOURCE_HSE_DIV2 ((uint32_t)0x00020300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV3 ((uint32_t)0x00030300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV4 ((uint32_t)0x00040300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV5 ((uint32_t)0x00050300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV6 ((uint32_t)0x00060300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV7 ((uint32_t)0x00070300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV8 ((uint32_t)0x00080300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV9 ((uint32_t)0x00090300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV10 ((uint32_t)0x000A0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV11 ((uint32_t)0x000B0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV12 ((uint32_t)0x000C0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV13 ((uint32_t)0x000D0300)***

- #define: ***RCC_RTCCLKSOURCE_HSE_DIV14 ((uint32_t)0x000E0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV15 ((uint32_t)0x000F0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV16 ((uint32_t)0x00100300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV17 ((uint32_t)0x00110300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV18 ((uint32_t)0x00120300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV19 ((uint32_t)0x00130300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV20 ((uint32_t)0x00140300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV21 ((uint32_t)0x00150300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV22 ((uint32_t)0x00160300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV23 ((uint32_t)0x00170300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV24 ((uint32_t)0x00180300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV25 ((uint32_t)0x00190300)***

- #define: ***RCC_RTCCLKSOURCE_HSE_DIV26 ((uint32_t)0x001A0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV27 ((uint32_t)0x001B0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV28 ((uint32_t)0x001C0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV29 ((uint32_t)0x001D0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV30 ((uint32_t)0x001E0300)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV31 ((uint32_t)0x001F0300)***

RCC_System_Clock_Source

- #define: ***RCC_SYSCLKSOURCE_HSI RCC_CFGR_SW_HSI***
- #define: ***RCC_SYSCLKSOURCE_HSE RCC_CFGR_SW_HSE***
- #define: ***RCC_SYSCLKSOURCE_PLLCLK RCC_CFGR_SW_PLL***

RCC_System_Clock_Type

- #define: ***RCC_CLOCKTYPE_SYSCLK ((uint32_t)0x00000001)***
- #define: ***RCC_CLOCKTYPE_HCLK ((uint32_t)0x00000002)***
- #define: ***RCC_CLOCKTYPE_PCLK1 ((uint32_t)0x00000004)***

- #define: ***RCC_CLOCKTYPE_PCLK2 ((uint32_t)0x00000008)***

37 HAL RCC Extension Driver

37.1 RCCEx Firmware driver registers structures

37.1.1 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 = 192 and Max_Data = 432. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAISpecifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between Min_Data = 192 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

37.1.2 RCC_PLLSAIInitTypeDef

RCC_PLLSAIInitTypeDef is defined in the stm32f4xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLLSAIN***
- ***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 = 192 and Max_Data = 432. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC
- ***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

37.1.3 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***

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***

37.2 RCCEx Firmware driver API description

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

37.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

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.

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***

37.2.2 Extended Peripheral Control functions

37.2.2.1 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 and RTC clocks).
Return values	• **HAL status**
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**
Parameters	• **PeriphClkInit :** pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(I2S and RTC clocks).

Return values	• **HAL status**
Notes	• A caution to be taken when HAL_RCCEx_PeriphCLKConfig() is used to select RTC clock selection, in this case the Reset of Backup domain will be applied in order to modify the RTC Clock source as consequence all backup domain (RTC and RCC_BDCR register expect BKPSRAM) will be reset • 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. • A caution to be taken when HAL_RCCEx_PeriphCLKConfig() is used to select RTC clock selection, in this case the Reset of Backup domain will be applied in order to modify the RTC Clock source as consequence all backup domain (RTC and RCC_BDCR register expect BKPSRAM) will be reset

37.2.2.2 HAL_RCCEx_GetPeriphCLKConfig

Function Name	**void HAL_RCCEx_GetPeriphCLKConfig (*RCC_PeriphCLKInitTypeDef ** PeriphClkInit)**
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	• **PeriphClkInit :** pointer to an RCC_PeriphCLKInitTypeDef structure that will be configured.
Return values	• None.
Notes	• None.

37.3 RCCEx Firmware driver defines

37.3.1 RCCEx

RCCEx

RCCEx_BitAddress_AliasRegion

- #define: ***PLLSAION_BitNumber 0x1C***

- #define: ***CR_PLLSAION_BB (PERIPH_BB_BASE + (RCC_CR_OFFSET * 32) + (PLLSAION_BitNumber * 4))***
- #define: ***RCC_DCKCFGR_OFFSET (RCC_OFFSET + 0x8C)***
- #define: ***TIMPRE_BitNumber 0x18***
- #define: ***DCKCFGR_TIMPRE_BB (PERIPH_BB_BASE + (RCC_DCKCFGR_OFFSET * 32) + (TIMPRE_BitNumber * 4))***

RCCEx_Periph_Clock_Selection

- #define: ***RCC_PERIPHCLK_I2S ((uint32_t)0x00000001)***
- #define: ***RCC_PERIPHCLK_SAI_PLLI2S ((uint32_t)0x00000002)***
- #define: ***RCC_PERIPHCLK_SAI_PLLSAI ((uint32_t)0x00000004)***
- #define: ***RCC_PERIPHCLK_LTDC ((uint32_t)0x00000008)***
- #define: ***RCC_PERIPHCLK_TIM ((uint32_t)0x00000010)***
- #define: ***RCC_PERIPHCLK_RTC ((uint32_t)0x00000020)***

RCCEx_PLLSAI_DIVR

- #define: ***RCC_PLLSAIDIVR_2 ((uint32_t)0x00000000)***

- #define: ***RCC_PLLSAIDIVR_4 ((uint32_t)0x00010000)***
- #define: ***RCC_PLLSAIDIVR_8 ((uint32_t)0x00020000)***
- #define: ***RCC_PLLSAIDIVR_16 ((uint32_t)0x00030000)***

RCCEx_SAI_BlockA_Clock_Source

- #define: ***RCC_SAIACLKSOURCE_PLLSAI ((uint32_t)0x00000000)***
- #define: ***RCC_SAIACLKSOURCE_PLLI2S ((uint32_t)0x00100000)***
- #define: ***RCC_SAIACLKSOURCE_EXT ((uint32_t)0x00200000)***

RCCEx_SAI_BlockB_Clock_Source

- #define: ***RCC_SAIBCLKSOURCE_PLLSAI ((uint32_t)0x00000000)***
- #define: ***RCC_SAIBCLKSOURCE_PLLI2S ((uint32_t)0x00400000)***
- #define: ***RCC_SAIBCLKSOURCE_EXT ((uint32_t)0x00800000)***

RCCEx_TIM_PRescaler_Selection

- #define: ***RCC_TIMPRES_DESACTIVATED ((uint8_t)0x00)***
- #define: ***RCC_TIMPRES_ACTIVATED ((uint8_t)0x01)***

38 HAL RNG Generic Driver

38.1 RNG Firmware driver registers structures

38.1.1 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***

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

38.1.2 RNG_TypeDef

RNG_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t SR***
- ***__IO uint32_t DR***

Field Documentation

- ***__IO uint32_t RNG_TypeDef::CR***
  - RNG control register, Address offset: 0x00
- ***__IO uint32_t RNG_TypeDef::SR***
  - RNG status register, Address offset: 0x04
- ***__IO uint32_t RNG_TypeDef::DR***
  - RNG data register, Address offset: 0x08

38.2 RNG Firmware driver API description

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

38.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __RNG_CLK_ENABLE() macro.
2. Activate the RNG peripheral using __HAL_RNG_ENABLE() macro.
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_GetRandomNumber() function.

38.2.2 Initialization and de-initialization 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
- ***HAL_RNG_Init()***
- ***HAL_RNG_DeInit()***
- ***HAL_RNG_MspInit()***
- ***HAL_RNG_MspDeInit()***

38.2.3 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
- ***HAL_RNG_GetRandomNumber()***
- ***HAL_RNG_GetRandomNumber_IT()***
- ***HAL_RNG_IRQHandler()***
- ***HAL_RNG_ReadyCallback()***
- ***HAL_RNG_ErrorCallback()***

38.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- ***HAL_RNG_GetState()***

38.2.5 Initialization and de-initialization functions

38.2.5.1 HAL_RNG_Init

Function Name	**HAL_StatusTypeDef HAL_RNG_Init (*RNG_HandleTypeDef ** hrng)**
Function Description	Initializes the RNG according to the specified parameters in the RNG_InitTypeDef and creates the associated handle.
Parameters	• **hrng :** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• **HAL status**
Notes	• None.

38.2.5.2 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**
Notes	• None.

38.2.5.3 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.
Notes	• None.

38.2.5.4 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.
Notes	• None.

38.2.6 Peripheral Control functions

38.2.6.1 HAL_RNG_GetRandomNumber

Function Name	**uint32_t HAL_RNG_GetRandomNumber (** ***RNG_HandleTypeDef **** **hrng)**
Function Description	Returns a 32-bit random number.
Parameters	• **hrng :** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• **32-bit random number**
Notes	• Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

38.2.6.2 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.
Parameters	• **hrng :** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• **32-bit random number**
Notes	• None.

38.2.6.3 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_FLAG(). 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_FLAG(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.

38.2.6.4 HAL_RNG_ReadyCallback

Function Name	**void HAL_RNG_ReadyCallback (** ***RNG_HandleTypeDef **** **hrng)**
Function Description	Data Ready callback in non-blocking mode.
Parameters	• **hrng :** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• None.
Notes	• None.

38.2.6.5 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.
Notes	• None.

38.2.7 Peripheral State functions

38.2.7.1 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**
Notes	• None.

38.3 RNG Firmware driver defines

38.3.1 RNG

RNG

RNG_Flag_definition

- #define: ***RNG_FLAG_DRDY ((uint32_t)0x0001)***

Data ready

- #define: ***RNG_FLAG_CECS ((uint32_t)0x0002)***

Clock error current status

- #define: ***RNG_FLAG_SECS ((uint32_t)0x0004)***

Seed error current status

RNG_Interrupt_definition

- #define: ***RNG_IT_CEI ((uint32_t)0x20)***

Clock error interrupt

- #define: ***RNG_IT_SEI ((uint32_t)0x40)***

Seed error interrupt

39 HAL RTC Generic Driver

39.1 RTC Firmware driver registers structures

39.1.1 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

39.1.2 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 = 0x7FFF
- ***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***

39.1.3 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

39.1.4 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32f4xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***

- ***uint8_t Seconds***
- ***uint32_t SubSeconds***
- ***uint8_t TimeFormat***
- ***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
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
  – Specifies the RTC Time SubSeconds. 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::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***

39.1.5 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***

39.1.6 RTC_TypeDef

RTC_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t TR***
- ***__IO uint32_t DR***
- ***__IO uint32_t CR***
- ***__IO uint32_t ISR***
- ***__IO uint32_t PRER***
- ***__IO uint32_t WUTR***
- ***__IO uint32_t CALIBR***
- ***__IO uint32_t ALRMAR***
- ***__IO uint32_t ALRMBR***
- ***__IO uint32_t WPR***
- ***__IO uint32_t SSR***
- ***__IO uint32_t SHIFTR***
- ***__IO uint32_t TSTR***
- ***__IO uint32_t TSDR***
- ***__IO uint32_t TSSSR***
- ***__IO uint32_t CALR***
- ***__IO uint32_t TAFCR***
- ***__IO uint32_t ALRMASSR***
- ***__IO uint32_t ALRMBSSR***
- ***uint32_t RESERVED7***
- ***__IO uint32_t BKP0R***
- ***__IO uint32_t BKP1R***
- ***__IO uint32_t BKP2R***
- ***__IO uint32_t BKP3R***
- ***__IO uint32_t BKP4R***
- ***__IO uint32_t BKP5R***
- ***__IO uint32_t BKP6R***

- ***__IO uint32_t BKP7R***
- ***__IO uint32_t BKP8R***
- ***__IO uint32_t BKP9R***
- ***__IO uint32_t BKP10R***
- ***__IO uint32_t BKP11R***
- ***__IO uint32_t BKP12R***
- ***__IO uint32_t BKP13R***
- ***__IO uint32_t BKP14R***
- ***__IO uint32_t BKP15R***
- ***__IO uint32_t BKP16R***
- ***__IO uint32_t BKP17R***
- ***__IO uint32_t BKP18R***
- ***__IO uint32_t BKP19R***

Field Documentation

- ***__IO uint32_t RTC_TypeDef::TR***
  - RTC time register, Address offset: 0x00
- ***__IO uint32_t RTC_TypeDef::DR***
  - RTC date register, Address offset: 0x04
- ***__IO uint32_t RTC_TypeDef::CR***
  - RTC control register, Address offset: 0x08
- ***__IO uint32_t RTC_TypeDef::ISR***
  - RTC initialization and status register, Address offset: 0x0C
- ***__IO uint32_t RTC_TypeDef::PRER***
  - RTC prescaler register, Address offset: 0x10
- ***__IO uint32_t RTC_TypeDef::WUTR***
  - RTC wakeup timer register, Address offset: 0x14
- ***__IO uint32_t RTC_TypeDef::CALIBR***
  - RTC calibration register, Address offset: 0x18
- ***__IO uint32_t RTC_TypeDef::ALRMAR***
  - RTC alarm A register, Address offset: 0x1C
- ***__IO uint32_t RTC_TypeDef::ALRMBR***
  - RTC alarm B register, Address offset: 0x20
- ***__IO uint32_t RTC_TypeDef::WPR***
  - RTC write protection register, Address offset: 0x24
- ***__IO uint32_t RTC_TypeDef::SSR***
  - RTC sub second register, Address offset: 0x28
- ***__IO uint32_t RTC_TypeDef::SHIFTR***
  - RTC shift control register, Address offset: 0x2C
- ***__IO uint32_t RTC_TypeDef::TSTR***
  - RTC time stamp time register, Address offset: 0x30
- ***__IO uint32_t RTC_TypeDef::TSDR***
  - RTC time stamp date register, Address offset: 0x34
- ***__IO uint32_t RTC_TypeDef::TSSSR***
  - RTC time-stamp sub second register, Address offset: 0x38
- ***__IO uint32_t RTC_TypeDef::CALR***
  - RTC calibration register, Address offset: 0x3C
- ***__IO uint32_t RTC_TypeDef::TAFCR***
  - RTC tamper and alternate function configuration register, Address offset: 0x40
- ***__IO uint32_t RTC_TypeDef::ALRMASSR***

  - RTC alarm A sub second register, Address offset: 0x44
- ***__IO uint32_t RTC_TypeDef::ALRMBSSR***
  - RTC alarm B sub second register, Address offset: 0x48
- ***uint32_t RTC_TypeDef::RESERVED7***
  - Reserved, 0x4C
- ***__IO uint32_t RTC_TypeDef::BKP0R***
  - RTC backup register 1, Address offset: 0x50
- ***__IO uint32_t RTC_TypeDef::BKP1R***
  - RTC backup register 1, Address offset: 0x54
- ***__IO uint32_t RTC_TypeDef::BKP2R***
  - RTC backup register 2, Address offset: 0x58
- ***__IO uint32_t RTC_TypeDef::BKP3R***
  - RTC backup register 3, Address offset: 0x5C
- ***__IO uint32_t RTC_TypeDef::BKP4R***
  - RTC backup register 4, Address offset: 0x60
- ***__IO uint32_t RTC_TypeDef::BKP5R***
  - RTC backup register 5, Address offset: 0x64
- ***__IO uint32_t RTC_TypeDef::BKP6R***
  - RTC backup register 6, Address offset: 0x68
- ***__IO uint32_t RTC_TypeDef::BKP7R***
  - RTC backup register 7, Address offset: 0x6C
- ***__IO uint32_t RTC_TypeDef::BKP8R***
  - RTC backup register 8, Address offset: 0x70
- ***__IO uint32_t RTC_TypeDef::BKP9R***
  - RTC backup register 9, Address offset: 0x74
- ***__IO uint32_t RTC_TypeDef::BKP10R***
  - RTC backup register 10, Address offset: 0x78
- ***__IO uint32_t RTC_TypeDef::BKP11R***
  - RTC backup register 11, Address offset: 0x7C
- ***__IO uint32_t RTC_TypeDef::BKP12R***
  - RTC backup register 12, Address offset: 0x80
- ***__IO uint32_t RTC_TypeDef::BKP13R***
  - RTC backup register 13, Address offset: 0x84
- ***__IO uint32_t RTC_TypeDef::BKP14R***
  - RTC backup register 14, Address offset: 0x88
- ***__IO uint32_t RTC_TypeDef::BKP15R***
  - RTC backup register 15, Address offset: 0x8C
- ***__IO uint32_t RTC_TypeDef::BKP16R***
  - RTC backup register 16, Address offset: 0x90
- ***__IO uint32_t RTC_TypeDef::BKP17R***
  - RTC backup register 17, Address offset: 0x94
- ***__IO uint32_t RTC_TypeDef::BKP18R***
  - RTC backup register 18, Address offset: 0x98
- ***__IO uint32_t RTC_TypeDef::BKP19R***
  - RTC backup register 19, Address offset: 0x9C

39.2 RTC Firmware driver API description

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

39.2.1 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

39.2.2 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.

39.2.3 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 __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.

39.2.4 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.

39.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

39.2.6 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 wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

- ***HAL_RTC_Init()***
- ***HAL_RTC_DeInit()***
- ***HAL_RTC_MspInit()***
- ***HAL_RTC_MspDeInit()***

39.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

- ***HAL_RTC_SetTime()***
- ***HAL_RTC_GetTime()***
- ***HAL_RTC_SetDate()***
- ***HAL_RTC_GetDate()***

39.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

- ***HAL_RTC_SetAlarm()***
- ***HAL_RTC_SetAlarm_IT()***
- ***HAL_RTC_DeactivateAlarm()***
- ***HAL_RTC_GetAlarm()***
- ***HAL_RTC_AlarmIRQHandler()***
- ***HAL_RTC_AlarmAEventCallback()***
- ***HAL_RTC_PollForAlarmAEvent()***

39.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- ***HAL_RTC_WaitForSynchro()***
- ***RTC_EnterInitMode()***
- ***RTC_ByteToBcd2()***
- ***RTC_Bcd2ToByte()***

39.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- ***HAL_RTC_GetState()***

39.2.11 Initialization and de-initialization functions

39.2.11.1 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**
Notes	• None.

39.2.11.2 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.

39.2.11.3 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.
Notes	• None.

39.2.11.4 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.
Notes	• None.

39.2.12 RTC Time and Date functions

39.2.12.1 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• None.

39.2.12.2 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• Call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers.

39.2.12.3 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• None.

39.2.12.4 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• None.

39.2.13 RTC Alarm functions

39.2.13.1 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• None.

39.2.13.2 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• None.

39.2.13.3 HAL_RTC_DeactivateAlarm

Function Name	**HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (** ***RTC_HandleTypeDef **** **hrtc, uint32_t Alarm)**
Function Description	Deactive 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**
Notes	• None.

39.2.13.4 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: – ***FORMAT_BIN :*** Binary data format – ***FORMAT_BCD :*** BCD data format
Return values	• **HAL status**
Notes	• None.

39.2.13.5 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.
Notes	• None.

39.2.13.6 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.
Notes	• None.

39.2.13.7 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**
Notes	• None.

39.2.14 Peripheral Control functions

39.2.14.1 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 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.

39.2.14.2 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.

39.2.14.3 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**
Notes	• None.

39.2.14.4 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**
Notes	• None.

39.2.15 Peripheral State functions

39.2.15.1 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**
Notes	• None.

39.3 RTC Firmware driver defines

39.3.1 RTC

RTC

RTC_AlarmDateWeekDay_Definitions

- #define: ***RTC_ALARMDATEWEEKDAYSEL_DATE ((uint32_t)0x00000000)***
- #define: ***RTC_ALARMDATEWEEKDAYSEL_WEEKDAY ((uint32_t)0x40000000)***

RTC_AlarmMask_Definitions

- #define: ***RTC_ALARMMASK_NONE ((uint32_t)0x00000000)***
- #define: ***RTC_ALARMMASK_DATEWEEKDAY RTC_ALRMAR_MSK4***
- #define: ***RTC_ALARMMASK_HOURS RTC_ALRMAR_MSK3***
- #define: ***RTC_ALARMMASK_MINUTES RTC_ALRMAR_MSK2***
- #define: ***RTC_ALARMMASK_SECONDS RTC_ALRMAR_MSK1***

- #define: ***RTC_ALARMMASK_ALL ((uint32_t)0x80808080)***

RTC_Alarms_Definitions

- #define: ***RTC_ALARM_A RTC_CR_ALRAE***

- #define: ***RTC_ALARM_B RTC_CR_ALRBE***

RTC_Alarm_Sub_Seconds_Masks_Definitions

- #define: ***RTC_ALARMSUBSECONDMASK_ALL ((uint32_t)0x00000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_1 ((uint32_t)0x01000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_2 ((uint32_t)0x02000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_3 ((uint32_t)0x03000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_4 ((uint32_t)0x04000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_5 ((uint32_t)0x05000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_6 ((uint32_t)0x06000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_7 ((uint32_t)0x07000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_8 ((uint32_t)0x08000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_9 ((uint32_t)0x09000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_10 ((uint32_t)0x0A000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_11 ((uint32_t)0x0B000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_12 ((uint32_t)0x0C000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14_13 ((uint32_t)0x0D000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_SS14 ((uint32_t)0x0E000000)***

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

- #define: ***RTC_ALARMSUBSECONDMASK_None ((uint32_t)0x0F000000)***

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

RTC_AM_PM_Definitions

- #define: ***RTC_HOURFORMAT12_AM ((uint8_t)0x00)***

- #define: ***RTC_HOURFORMAT12_PM ((uint8_t)0x40)***

RTC_DayLightSaving_Definitions

- #define: ***RTC_DAYLIGHTSAVING_SUB1H ((uint32_t)0x00020000)***

- #define: ***RTC_DAYLIGHTSAVING_ADD1H ((uint32_t)0x00010000)***

- #define: ***RTC_DAYLIGHTSAVING_NONE ((uint32_t)0x00000000)***

RTC_Exported_Constants

- #define: ***RTC_TR_RESERVED_MASK ((uint32_t)0x007F7F7F)***
- #define: ***RTC_DR_RESERVED_MASK ((uint32_t)0x00FFFF3F)***
- #define: ***RTC_INIT_MASK ((uint32_t)0xFFFFFFFF)***
- #define: ***RTC_RSF_MASK ((uint32_t)0xFFFFFF5F)***
- #define: ***RTC_FLAGS_MASK ((uint32_t)(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_WUTWF | \ RTC_FLAG_ALRBWF | RTC_FLAG_ALRAWF | RTC_FLAG_TAMP1F | \ RTC_FLAG_RECALPF | RTC_FLAG_SHPF))***
- #define: ***RTC_TIMEOUT_VALUE 1000***

RTC_Flags_Definitions

- #define: ***RTC_FLAG_RECALPF ((uint32_t)0x00010000)***
- #define: ***RTC_FLAG_TAMP2F ((uint32_t)0x00004000)***
- #define: ***RTC_FLAG_TAMP1F ((uint32_t)0x00002000)***
- #define: ***RTC_FLAG_TSOVF ((uint32_t)0x00001000)***

- #define: ***RTC_FLAG_TSF ((uint32_t)0x00000800)***
- #define: ***RTC_FLAG_WUTF ((uint32_t)0x00000400)***
- #define: ***RTC_FLAG_ALRBF ((uint32_t)0x00000200)***
- #define: ***RTC_FLAG_ALRAF ((uint32_t)0x00000100)***
- #define: ***RTC_FLAG_INITF ((uint32_t)0x00000040)***
- #define: ***RTC_FLAG_RSF ((uint32_t)0x00000020)***
- #define: ***RTC_FLAG_INITS ((uint32_t)0x00000010)***
- #define: ***RTC_FLAG_SHPF ((uint32_t)0x00000008)***
- #define: ***RTC_FLAG_WUTWF ((uint32_t)0x00000004)***
- #define: ***RTC_FLAG_ALRBWF ((uint32_t)0x00000002)***
- #define: ***RTC_FLAG_ALRAWF ((uint32_t)0x00000001)***

RTC_Hour_Formats

- #define: ***RTC_HOURFORMAT_24 ((uint32_t)0x00000000)***
- #define: ***RTC_HOURFORMAT_12 ((uint32_t)0x00000040)***

RTC_Input_parameter_format_definitions

- #define: ***FORMAT_BIN ((uint32_t)0x000000000)***
- #define: ***FORMAT_BCD ((uint32_t)0x000000001)***

RTC_Interrupts_Definitions

- #define: ***RTC_IT_TS ((uint32_t)0x00008000)***
- #define: ***RTC_IT_WUT ((uint32_t)0x00004000)***
- #define: ***RTC_IT_ALRB ((uint32_t)0x00002000)***
- #define: ***RTC_IT_ALRA ((uint32_t)0x00001000)***
- #define: ***RTC_IT_TAMP ((uint32_t)0x00000004)***
- #define: ***RTC_IT_TAMP1 ((uint32_t)0x00020000)***
- #define: ***RTC_IT_TAMP2 ((uint32_t)0x00040000)***

RTC_Month_Date_Definitions

- #define: ***RTC_MONTH_JANUARY ((uint8_t)0x01)***
- #define: ***RTC_MONTH_FEBRUARY ((uint8_t)0x02)***
- #define: ***RTC_MONTH_MARCH ((uint8_t)0x03)***
- #define: ***RTC_MONTH_APRIL ((uint8_t)0x04)***
- #define: ***RTC_MONTH_MAY ((uint8_t)0x05)***
- #define: ***RTC_MONTH_JUNE ((uint8_t)0x06)***
- #define: ***RTC_MONTH_JULY ((uint8_t)0x07)***
- #define: ***RTC_MONTH_AUGUST ((uint8_t)0x08)***
- #define: ***RTC_MONTH_SEPTEMBER ((uint8_t)0x09)***
- #define: ***RTC_MONTH_OCTOBER ((uint8_t)0x10)***
- #define: ***RTC_MONTH_NOVEMBER ((uint8_t)0x11)***
- #define: ***RTC_MONTH_DECEMBER ((uint8_t)0x12)***

RTC_Output_Polarity_Definitions

- #define: ***RTC_OUTPUT_POLARITY_HIGH ((uint32_t)0x00000000)***
- #define: ***RTC_OUTPUT_POLARITY_LOW ((uint32_t)0x00100000)***

RTC_Output_selection_Definitions

- #define: ***RTC_OUTPUT_DISABLE ((uint32_t)0x00000000)***
- #define: ***RTC_OUTPUT_ALARMA ((uint32_t)0x00200000)***
- #define: ***RTC_OUTPUT_ALARMB ((uint32_t)0x00400000)***
- #define: ***RTC_OUTPUT_WAKEUP ((uint32_t)0x00600000)***

RTC_Output_Type_ALARM_OUT

- #define: ***RTC_OUTPUT_TYPE_OPENDRAIN ((uint32_t)0x00000000)***
- #define: ***RTC_OUTPUT_TYPE_PUSHPULL ((uint32_t)0x00040000)***

RTC_StoreOperation_Definitions

- #define: ***RTC_STOREOPERATION_RESET ((uint32_t)0x00000000)***
- #define: ***RTC_STOREOPERATION_SET ((uint32_t)0x00040000)***

RTC_WeekDay_Definitions

- #define: ***RTC_WEEKDAY_MONDAY ((uint8_t)0x01)***

- #define: ***RTC_WEEKDAY_TUESDAY ((uint8_t)0x02)***
- #define: ***RTC_WEEKDAY_WEDNESDAY ((uint8_t)0x03)***
- #define: ***RTC_WEEKDAY_THURSDAY ((uint8_t)0x04)***
- #define: ***RTC_WEEKDAY_FRIDAY ((uint8_t)0x05)***
- #define: ***RTC_WEEKDAY_SATURDAY ((uint8_t)0x06)***
- #define: ***RTC_WEEKDAY_SUNDAY ((uint8_t)0x07)***

40 HAL RTC Extension Driver

40.1 RTCEx Firmware driver registers structures

40.1.1 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***

40.2 RTCEx Firmware driver API description

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

40.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTC_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer in interrupt mode using the HAL_RTC_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTC_GetWakeUpTimer() function.

TimeStamp configuration

- Configure the RTC_AFx trigger and enable the RTC TimeStamp using the HAL_RTC_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTC_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTC_GetTimeStamp() function.
- The TIMESTAMP alternate function can be mapped either to RTC_AF1 (PC13) or RTC_AF2 (PI8) depending on the value of TSINSEL bit in RTC_TAFCR register. The corresponding pin is also selected by HAL_RTC_SetTimeStamp() or HAL_RTC_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_RTC_SetTamper() function. You can configure RTC Tamper in interrupt mode using HAL_RTC_SetTamper_IT() function.
- The TAMPER1 alternate function can be mapped either to RTC_AF1 (PC13) or RTC_AF2 (PI8) depending on the value of TAMP1INSEL bit in RTC_TAFCR register. The corresponding pin is also selected by HAL_RTC_SetTamper() or HAL_RTC_SetTamper_IT() function.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTC_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTC_BKUPRead() function.

40.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

- *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()*

40.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

40.2.4 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.
- *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()*

40.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alram B callback
- RTC Poll for Alarm B request
- *HAL_RTCEx_AlarmBEventCallback()*
- *HAL_RTCEx_PollForAlarmBEvent()*

40.2.6 RTC TimeStamp and Tamper functions

40.2.6.1 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_PC13 :*** PC13 is selected as RTC TimeStamp Pin. – ***RTC_TIMESTAMPPIN_PI8 :*** PI8 is selected as RTC TimeStamp Pin.
Return values	• **HAL status**
Notes	• This API must be called before enabling the TimeStamp feature.

40.2.6.2 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.
Parameters	• **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_PC13 :*** PC13 is selected as RTC TimeStamp Pin. – ***RTC_TIMESTAMPPIN_PI8 :*** PI8 is selected as RTC TimeStamp Pin.
Return values	• **HAL status**
Notes	• This API must be called before enabling the TimeStamp feature.

40.2.6.3 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**
Notes	• None.

40.2.6.4 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: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	• **HAL status**
Notes	• None.

40.2.6.5 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.

40.2.6.6 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.

40.2.6.7 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**
Notes	• None.

40.2.6.8 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.
Notes	• None.

40.2.6.9 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.
Notes	• None.

40.2.6.10 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.
Notes	• None.

40.2.6.11 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.
Notes	• None.

40.2.6.12 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**
Notes	• None.

40.2.6.13 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**
Notes	• None.

40.2.6.14 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**
Notes	• None.

40.2.7 RTC Wake-up functions

40.2.7.1 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**
Notes	• None.

40.2.7.2 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**
Notes	• None.

40.2.7.3 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**
Notes	• None.

40.2.7.4 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**
Notes	• None.

40.2.7.5 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	• None.

40.2.7.6 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.
Notes	• None.

40.2.7.7 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**
Notes	• None.

40.2.8 Extension Peripheral Control functions

40.2.8.1 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.
Notes	• None.

40.2.8.2 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**
Notes	• None.

40.2.8.3 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.

40.2.8.4 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**
Notes	• None.

40.2.8.5 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 periode is 32s. – ***RTC_SMOOTHCALIB_PERIOD_16SEC :*** The smooth calibration periode is 16s. – ***RTC_SMOOTHCALIB_PERIOD_8SEC :*** The smooth calibartion periode 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 puls every 2*11 pulses.

	– ***RTC_SMOOTHCALIB_PLUSPULSES_RESET :*** No RTCCLK pulses are added. • **SmouthCalibMinusPulsesValue :** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	• **HAL status**
Notes	• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmouthCalibMinusPulsesValue must be equal to 0.

40.2.8.6 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.

40.2.8.7 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**
Notes	• None.

40.2.8.8 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**
Notes	• None.

40.2.8.9 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**
Notes	• None.

40.2.8.10 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**
Notes	• None.

40.2.8.11 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.

40.2.8.12 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.

40.2.9 Extended features functions

40.2.9.1 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.
Notes	• None.

40.2.9.2 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**
Notes	• None.

40.3 RTCEx Firmware driver defines

40.3.1 RTCEx

RTCEx

RTCEx_Add_1_Second_Parameter_Definitions

- #define: ***RTC_SHIFTADD1S_RESET ((uint32_t)0x00000000)***
- #define: ***RTC_SHIFTADD1S_SET ((uint32_t)0x80000000)***

RTCEx_Backup_Registers_Definitions

- #define: ***RTC_BKP_DR0 ((uint32_t)0x00000000)***
- #define: ***RTC_BKP_DR1 ((uint32_t)0x00000001)***
- #define: ***RTC_BKP_DR2 ((uint32_t)0x00000002)***
- #define: ***RTC_BKP_DR3 ((uint32_t)0x00000003)***
- #define: ***RTC_BKP_DR4 ((uint32_t)0x00000004)***
- #define: ***RTC_BKP_DR5 ((uint32_t)0x00000005)***
- #define: ***RTC_BKP_DR6 ((uint32_t)0x00000006)***
- #define: ***RTC_BKP_DR7 ((uint32_t)0x00000007)***
- #define: ***RTC_BKP_DR8 ((uint32_t)0x00000008)***
- #define: ***RTC_BKP_DR9 ((uint32_t)0x00000009)***

- #define: ***RTC_BKP_DR10 ((uint32_t)0x0000000A)***
- #define: ***RTC_BKP_DR11 ((uint32_t)0x0000000B)***
- #define: ***RTC_BKP_DR12 ((uint32_t)0x0000000C)***
- #define: ***RTC_BKP_DR13 ((uint32_t)0x0000000D)***
- #define: ***RTC_BKP_DR14 ((uint32_t)0x0000000E)***
- #define: ***RTC_BKP_DR15 ((uint32_t)0x0000000F)***
- #define: ***RTC_BKP_DR16 ((uint32_t)0x00000010)***
- #define: ***RTC_BKP_DR17 ((uint32_t)0x00000011)***
- #define: ***RTC_BKP_DR18 ((uint32_t)0x00000012)***
- #define: ***RTC_BKP_DR19 ((uint32_t)0x00000013)***

RTCEx_Calib_Output_selection_Definitions

- #define: ***RTC_CALIBOUTPUT_512HZ ((uint32_t)0x00000000)***
- #define: ***RTC_CALIBOUTPUT_1HZ ((uint32_t)0x00080000)***

RTCEx_Digital_Calibration_Definitions

- #define: ***RTC_CALIBSIGN_POSITIVE ((uint32_t)0x00000000)***

- #define: ***RTC_CALIBSIGN_NEGATIVE ((uint32_t)0x00000080)***

RTCEx_Smooth_calib_period_Definitions

- #define: ***RTC_SMOOTHCALIB_PERIOD_32SEC ((uint32_t)0x00000000)***

If RTCCLK = 32768 Hz, Smooth calibation period is 32s, else 2exp20 RTCCLK seconds

- #define: ***RTC_SMOOTHCALIB_PERIOD_16SEC ((uint32_t)0x00002000)***

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

- #define: ***RTC_SMOOTHCALIB_PERIOD_8SEC ((uint32_t)0x00004000)***

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

RTCEx_Smooth_calib_Plus_pulses_Definitions

- #define: ***RTC_SMOOTHCALIB_PLUSPULSES_SET ((uint32_t)0x00008000)***

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

- #define: ***RTC_SMOOTHCALIB_PLUSPULSES_RESET ((uint32_t)0x00000000)***

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

RTCEx_Tamper_Filter_Definitions

- #define: ***RTC_TAMPERFILTER_DISABLE ((uint32_t)0x00000000)***

Tamper filter is disabled

- #define: ***RTC_TAMPERFILTER_2SAMPLE ((uint32_t)0x00000800)***

Tamper is activated after 2 consecutive samples at the active level

- #define: ***RTC_TAMPERFILTER_4SAMPLE ((uint32_t)0x00001000)***

Tamper is activated after 4 consecutive samples at the active level

- #define: ***RTC_TAMPERFILTER_8SAMPLE ((uint32_t)0x00001800)***

Tamper is activated after 8 consecutive samples at the active leve.

RTCEx_Tamper_Pins_Definitions

- #define: ***RTC_TAMPER_1 RTC_TAFCR_TAMP1E***

- #define: ***RTC_TAMPER_2 RTC_TAFCR_TAMP2E***

RTCEx_Tamper_Pins_Selection

- #define: ***RTC_TAMPERPIN_PC13 ((uint32_t)0x00000000)***

- #define: ***RTC_TAMPERPIN_PI8 ((uint32_t)0x00010000)***

RTCEx_Tamper_Pin_Precharge_Duration_Definitions

- #define: ***RTC_TAMPERPRECHARGEDURATION_1RTCCLK ((uint32_t)0x00000000)***

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

- #define: ***RTC_TAMPERPRECHARGEDURATION_2RTCCLK ((uint32_t)0x00002000)***

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

- #define: ***RTC_TAMPERPRECHARGEDURATION_4RTCCLK ((uint32_t)0x00004000)***

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

- #define: ***RTC_TAMPERPRECHARGEDURATION_8RTCCLK ((uint32_t)0x00006000)***

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx_Tamper_Pull_UP_Definitions

- #define: ***RTC_TAMPER_PULLUP_ENABLE ((uint32_t)0x00000000)***

TimeStamp on Tamper Detection event saved

- #define: ***RTC_TAMPER_PULLUP_DISABLE ((uint32_t)RTC_TAFCR_TAMPPUDIS)***

TimeStamp on Tamper Detection event is not saved

RTCEx_Tamper_Sampling_Frequencies_Definitions

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768 ((uint32_t)0x00000000)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384 ((uint32_t)0x00000100)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192 ((uint32_t)0x00000200)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096 ((uint32_t)0x00000300)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048 ((uint32_t)0x00000400)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024 ((uint32_t)0x00000500)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512 ((uint32_t)0x00000600)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

- #define: ***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256 ((uint32_t)0x00000700)***

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTCEx_Tamper_TimeStampOnTamperDetection_Definitions

- #define: ***RTC_TIMESTAMPONTAMPERDETECTION_ENABLE ((uint32_t)RTC_TAFCR_TAMPTS)***

TimeStamp on Tamper Detection event saved

- #define: ***RTC_TIMESTAMPONTAMPERDETECTION_DISABLE ((uint32_t)0x00000000)***

TimeStamp on Tamper Detection event is not saved

RTCEx_Tamper_Trigger_Definitions

- #define: ***RTC_TAMPERTRIGGER_RISINGEDGE ((uint32_t)0x00000000)***
- #define: ***RTC_TAMPERTRIGGER_FALLINGEDGE ((uint32_t)0x00000002)***
- #define: ***RTC_TAMPERTRIGGER_LOWLEVEL RTC_TAMPERTRIGGER_RISINGEDGE***
- #define: ***RTC_TAMPERTRIGGER_HIGHLEVEL RTC_TAMPERTRIGGER_FALLINGEDGE***

RTCEx_TimeStamp_Pin_Selection

- #define: ***RTC_TIMESTAMPPIN_PC13 ((uint32_t)0x00000000)***
- #define: ***RTC_TIMESTAMPPIN_PI8 ((uint32_t)0x00020000)***

RTCEx_Time_Stamp_Edges_definitions

- #define: ***RTC_TIMESTAMPEDGE_RISING ((uint32_t)0x00000000)***
- #define: ***RTC_TIMESTAMPEDGE_FALLING ((uint32_t)0x00000008)***

RTCEx_Wakeup_Timer_Definitions

- #define: ***RTC_WAKEUPCLOCK_RTCCLK_DIV16 ((uint32_t)0x00000000)***

- #define: ***RTC_WAKEUPCLOCK_RTCCLK_DIV8 ((uint32_t)0x00000001)***
- #define: ***RTC_WAKEUPCLOCK_RTCCLK_DIV4 ((uint32_t)0x00000002)***
- #define: ***RTC_WAKEUPCLOCK_RTCCLK_DIV2 ((uint32_t)0x00000003)***
- #define: ***RTC_WAKEUPCLOCK_CK_SPRE_16BITS ((uint32_t)0x00000004)***
- #define: ***RTC_WAKEUPCLOCK_CK_SPRE_17BITS ((uint32_t)0x00000006)***

41 HAL SAI Generic Driver

41.1 SAI Firmware driver registers structures

41.1.1 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***
- ***uint16_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint16_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***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
- ***uint16_t* SAI_HandleTypeDef::pTxBuffPtr***
  - Pointer to SAI Tx transfer Buffer
- ***uint16_t SAI_HandleTypeDef::TxXferSize***
  - SAI Tx transfer size
- ***uint16_t SAI_HandleTypeDef::TxXferCount***
  - SAI Tx transfer counter
- ***uint16_t* SAI_HandleTypeDef::pRxBuffPtr***
  - Pointer to SAI Rx transfer buffer
- ***uint16_t SAI_HandleTypeDef::RxXferSize***
  - SAI Rx transfer size
- ***uint16_t SAI_HandleTypeDef::RxXferCount***
  - SAI Rx transfer counter
- ***DMA_HandleTypeDef* SAI_HandleTypeDef::hdmatx***

  - SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* SAI_HandleTypeDef::hdmarx***
  - SAI Rx DMA handle parameters
- ***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

41.1.2 SAI_InitTypeDef

SAI_InitTypeDef is defined in the stm32f4xx_hal_sai.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t AudioMode***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***
- ***uint32_t Synchro***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t ClockSource***
- ***uint32_t AudioFrequency***

Field Documentation

- ***uint32_t SAI_InitTypeDef::Protocol***
  - Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***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::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***
- ***uint32_t SAI_InitTypeDef::Synchro***
  - Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***
- ***uint32_t SAI_InitTypeDef::OutputDrive***

  - Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***
- ***uint32_t SAI_InitTypeDef::NoDivider***
  - Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***
- ***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 can be a value of ***SAI_Clock_Source***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***
  - Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***

41.1.3 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. : 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***

41.1.4 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***

41.1.5 SAI_Block_TypeDef

SAI_Block_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t FRCR***
- ***__IO uint32_t SLOTR***
- ***__IO uint32_t IMR***
- ***__IO uint32_t SR***
- ***__IO uint32_t CLRFR***
- ***__IO uint32_t DR***

Field Documentation

- ***__IO uint32_t SAI_Block_TypeDef::CR1***
  - SAI block x configuration register 1, Address offset: 0x04
- ***__IO uint32_t SAI_Block_TypeDef::CR2***
  - SAI block x configuration register 2, Address offset: 0x08
- ***__IO uint32_t SAI_Block_TypeDef::FRCR***
  - SAI block x frame configuration register, Address offset: 0x0C

- ***__IO uint32_t SAI_Block_TypeDef::SLOTR***
  - SAI block x slot register, Address offset: 0x10
- ***__IO uint32_t SAI_Block_TypeDef::IMR***
  - SAI block x interrupt mask register, Address offset: 0x14
- ***__IO uint32_t SAI_Block_TypeDef::SR***
  - SAI block x status register, Address offset: 0x18
- ***__IO uint32_t SAI_Block_TypeDef::CLRFR***
  - SAI block x clear flag register, Address offset: 0x1C
- ***__IO uint32_t SAI_Block_TypeDef::DR***
  - SAI block x data register, Address offset: 0x20

41.1.6 SAI_TypeDef

SAI_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t GCR***

Field Documentation

- ***__IO uint32_t SAI_TypeDef::GCR***
  - SAI global configuration register, Address offset: 0x00

41.2 SAI Firmware driver API description

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

41.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure.
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. Program the SAI Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_SAI_Init() function. The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __SAI_ENABLE_IT() and __SAI_DISABLE_IT() inside the transmit and receive process.

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.

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.

In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

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 transfer 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 transfer 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 macros list

Below the list of most used macros in USART 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

41.2.2 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.
- ***HAL_SAI_Init()***
- ***HAL_SAI_DeInit()***
- ***HAL_SAI_MspInit()***

- *HAL_SAI_MspDeInit()*

41.2.3 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()
  - HAL_SAI_TransmitReceive()
- Non Blocking mode functions with Interrupt are :
  - HAL_SAI_Transmit_IT()
  - HAL_SAI_Receive_IT()
  - HAL_SAI_TransmitReceive_IT()
- Non Blocking mode functions with DMA are :
  - HAL_SAI_Transmit_DMA()
  - HAL_SAI_Receive_DMA()
  - HAL_SAI_TransmitReceive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SAI_TxCpltCallback()
  - HAL_SAI_RxCpltCallback()
  - HAL_SAI_ErrorCallback()
- *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_Transmit_DMA()*
- *HAL_SAI_Receive_DMA()*
- *HAL_SAI_IRQHandler()*
- *HAL_SAI_TxCpltCallback()*
- *HAL_SAI_TxHalfCpltCallback()*
- *HAL_SAI_RxCpltCallback()*
- *HAL_SAI_RxHalfCpltCallback()*
- *HAL_SAI_ErrorCallback()*

41.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_SAI_GetState()*
- *HAL_SAI_GetError()*

41.2.5 Initialization and de-initialization functions

41.2.5.1 HAL_SAI_Init

Function Name	**HAL_StatusTypeDef HAL_SAI_Init (*SAI_HandleTypeDef* * hsai)**
Function Description	Initializes the SAI according to the specified parameters in the SAI_InitTypeDef and create the associated handle.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL status**
Notes	• None.

41.2.5.2 HAL_SAI_DeInit

Function Name	**HAL_StatusTypeDef HAL_SAI_DeInit (*SAI_HandleTypeDef* * hsai)**
Function Description	DeInitializes the SAI peripheral.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL status**
Notes	• None.

41.2.5.3 HAL_SAI_MspInit

Function Name	**void HAL_SAI_MspInit (*SAI_HandleTypeDef* * hsai)**
Function Description	SAI MSP Init.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• None.
Notes	• None.

41.2.5.4 HAL_SAI_MspDeInit

Function Name	**void HAL_SAI_MspDeInit (*SAI_HandleTypeDef ** hsai)**
Function Description	SAI MSP DeInit.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• None.
Notes	• None.

41.2.6 IO operation functions

41.2.6.1 HAL_SAI_Transmit

Function Name	**HAL_StatusTypeDef HAL_SAI_Transmit (*SAI_HandleTypeDef ** hsai, uint16_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Transmits 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**
Notes	• None.

41.2.6.2 HAL_SAI_Receive

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

Function Description	Receives 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**
Notes	• None.

41.2.6.3 HAL_SAI_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_SAI_Transmit_IT (*SAI_HandleTypeDef ** hsai, uint16_t * pData, uint16_t Size)**
Function Description	Transmits an amount of data in no-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**
Notes	• None.

41.2.6.4 HAL_SAI_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_SAI_Receive_IT (*SAI_HandleTypeDef ** hsai, uint16_t * pData, uint16_t Size)**
Function Description	Receives an amount of data in no-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**
Notes	• None.

41.2.6.5 HAL_SAI_DMAPause

Function Name	**HAL_StatusTypeDef HAL_SAI_DMAPause (*SAI_HandleTypeDef * * hsai)**
Function Description	Pauses 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**
Notes	• None.

41.2.6.6 HAL_SAI_DMAResume

Function Name	**HAL_StatusTypeDef HAL_SAI_DMAResume (*SAI_HandleTypeDef * * hsai)**
Function Description	Resumes 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**
Notes	• None.

41.2.6.7 HAL_SAI_DMAStop

Function Name	**HAL_StatusTypeDef HAL_SAI_DMAStop (*SAI_HandleTypeDef * * hsai)**
Function Description	Stops 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**
Notes	• None.

41.2.6.8 HAL_SAI_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_SAI_Transmit_DMA (*SAI_HandleTypeDef ** hsai, uint16_t * pData, uint16_t Size)**
Function Description	Transmits an amount of data in no-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**
Notes	• None.

41.2.6.9 HAL_SAI_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_SAI_Receive_DMA (*SAI_HandleTypeDef ** hsai, uint16_t * pData, uint16_t Size)**
Function Description	Receives an amount of data in no-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**
Notes	• None.

41.2.6.10 HAL_SAI_IRQHandler

Function Name	**void HAL_SAI_IRQHandler (*SAI_HandleTypeDef ** hsai)**
Function Description	This function handles SAI interrupt request.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that

	contains the configuration information for SAI module.
Return values	• **HAL status**
Notes	• None.

41.2.6.11 HAL_SAI_TxCpltCallback

Function Name	**void HAL_SAI_TxCpltCallback (*SAI_HandleTypeDef ** hsai)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• None.
Notes	• None.

41.2.6.12 HAL_SAI_TxHalfCpltCallback

Function Name	**void HAL_SAI_TxHalfCpltCallback (*SAI_HandleTypeDef ** hsai)**
Function Description	Tx Transfer Half completed callbacks.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• None.
Notes	• None.

41.2.6.13 HAL_SAI_RxCpltCallback

Function Name	**void HAL_SAI_RxCpltCallback (*SAI_HandleTypeDef ** hsai)**
Function Description	Rx Transfer completed callbacks.

Parameters	• **hdma :** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None.
Notes	• None.

41.2.6.14 HAL_SAI_RxHalfCpltCallback

Function Name	**void HAL_SAI_RxHalfCpltCallback (*SAI_HandleTypeDef ** hsai)**
Function Description	Rx Transfer half completed callbacks.
Parameters	• **hdma :** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None.
Notes	• None.

41.2.6.15 HAL_SAI_ErrorCallback

Function Name	**void HAL_SAI_ErrorCallback (*SAI_HandleTypeDef ** hsai)**
Function Description	SAI error callbacks.
Parameters	• **hdma :** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None.
Notes	• None.

41.2.7 Peripheral State functions

41.2.7.1 HAL_SAI_GetState

Function Name	**HAL_SAI_StateTypeDef HAL_SAI_GetState (*SAI_HandleTypeDef* * hsai)**
Function Description	Returns the SAI state.
Parameters	• **hsai :** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	• **HAL state**
Notes	• None.

41.2.7.2 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**
Notes	• None.

41.3 SAI Firmware driver defines

41.3.1 SAI

SAI

SAI_Audio_Frequency

- #define: ***SAI_AUDIO_FREQUENCY_192K ((uint32_t)192000)***
- #define: ***SAI_AUDIO_FREQUENCY_96K ((uint32_t)96000)***

- #define: ***SAI_AUDIO_FREQUENCY_48K ((uint32_t)48000)***
- #define: ***SAI_AUDIO_FREQUENCY_44K ((uint32_t)44100)***
- #define: ***SAI_AUDIO_FREQUENCY_32K ((uint32_t)32000)***
- #define: ***SAI_AUDIO_FREQUENCY_22K ((uint32_t)22050)***
- #define: ***SAI_AUDIO_FREQUENCY_16K ((uint32_t)16000)***
- #define: ***SAI_AUDIO_FREQUENCY_11K ((uint32_t)11025)***
- #define: ***SAI_AUDIO_FREQUENCY_8K ((uint32_t)8000)***

SAI_Block_Clock_Strobing

- #define: ***SAI_CLOCKSTROBING_FALLINGEDGE ((uint32_t)0x00000000)***
- #define: ***SAI_CLOCKSTROBING_RISINGEDGE ((uint32_t)SAI_xCR1_CKSTR)***

SAI_Block_Companding_Mode

- #define: ***SAI_NOCOMPANDING ((uint32_t)0x00000000)***
- #define: ***SAI_ULAW_1CPL_COMPANDING ((uint32_t)0x00008000)***

- #define: ***SAI_ALAW_1CPL_COMPANDING ((uint32_t)0x0000C000)***
- #define: ***SAI_ULAW_2CPL_COMPANDING ((uint32_t)0x0000A000)***
- #define: ***SAI_ALAW_2CPL_COMPANDING ((uint32_t)0x0000E000)***

SAI_Block_Data_Size

- #define: ***SAI_DATASIZE_8 ((uint32_t)0x00000040)***
- #define: ***SAI_DATASIZE_10 ((uint32_t)0x00000060)***
- #define: ***SAI_DATASIZE_16 ((uint32_t)0x00000080)***
- #define: ***SAI_DATASIZE_20 ((uint32_t)0x000000A0)***
- #define: ***SAI_DATASIZE_24 ((uint32_t)0x000000C0)***
- #define: ***SAI_DATASIZE_32 ((uint32_t)0x000000E0)***

SAI_Block_Fifo_Status_Level

- #define: ***SAI_FIFOStatus_Empty ((uint32_t)0x00000000)***
- #define: ***SAI_FIFOStatus_Less1QuarterFull ((uint32_t)0x00010000)***
- #define: ***SAI_FIFOStatus_1QuarterFull ((uint32_t)0x00020000)***

- #define: ***SAI_FIFOStatus_HalfFull ((uint32_t)0x00030000)***

- #define: ***SAI_FIFOStatus_3QuartersFull ((uint32_t)0x00040000)***

- #define: ***SAI_FIFOStatus_Full ((uint32_t)0x00050000)***

SAI_Block_Fifo_Threshold

- #define: ***SAI_FIFOTHRESHOLD_EMPTY ((uint32_t)0x00000000)***

- #define: ***SAI_FIFOTHRESHOLD_1QF ((uint32_t)0x00000001)***

- #define: ***SAI_FIFOTHRESHOLD_HF ((uint32_t)0x00000002)***

- #define: ***SAI_FIFOTHRESHOLD_3QF ((uint32_t)0x00000003)***

- #define: ***SAI_FIFOTHRESHOLD_FULL ((uint32_t)0x00000004)***

SAI_Block_Flags_Definition

- #define: ***SAI_FLAG_OVRUDR ((uint32_t)SAI_xSR_OVRUDR)***

- #define: ***SAI_FLAG_MUTEDET ((uint32_t)SAI_xSR_MUTEDET)***

- #define: ***SAI_FLAG_WCKCFG ((uint32_t)SAI_xSR_WCKCFG)***

- #define: ***SAI_FLAG_FREQ ((uint32_t)SAI_xSR_FREQ)***
- #define: ***SAI_FLAG_CNRDY ((uint32_t)SAI_xSR_CNRDY)***
- #define: ***SAI_FLAG_AFSDET ((uint32_t)SAI_xSR_AFSDET)***
- #define: ***SAI_FLAG_LFSDET ((uint32_t)SAI_xSR_LFSDET)***

SAI_Block_FS_Definition

- #define: ***SAI_FS_STARTFRAME ((uint32_t)0x00000000)***
- #define: ***SAI_FS_CHANNEL_IDENTIFICATION ((uint32_t)SAI_xFRCR_FSDEF)***

SAI_Block_FS_Offset

- #define: ***SAI_FS_FIRSTBIT ((uint32_t)0x00000000)***
- #define: ***SAI_FS_BEFOREFIRSTBIT ((uint32_t)SAI_xFRCR_FSOFF)***

SAI_Block_FS_Polarity

- #define: ***SAI_FS_ACTIVE_LOW ((uint32_t)0x00000000)***
- #define: ***SAI_FS_ACTIVE_HIGH ((uint32_t)SAI_xFRCR_FSPO)***

SAI_Block_Interrupts_Definition

- #define: ***SAI_IT_OVRUDR ((uint32_t)SAI_xIMR_OVRUDRIE)***

- #define: ***SAI_IT_MUTEDET ((uint32_t)SAI_xIMR_MUTEDETIE)***
- #define: ***SAI_IT_WCKCFG ((uint32_t)SAI_xIMR_WCKCFGIE)***
- #define: ***SAI_IT_FREQ ((uint32_t)SAI_xIMR_FREQIE)***
- #define: ***SAI_IT_CNRDY ((uint32_t)SAI_xIMR_CNRDYIE)***
- #define: ***SAI_IT_AFSDET ((uint32_t)SAI_xIMR_AFSDETIE)***
- #define: ***SAI_IT_LFSDET ((uint32_t)SAI_xIMR_LFSDETIE)***

SAI_Block_Mode

- #define: ***SAI_MODEMASTER_TX ((uint32_t)0x00000000)***
- #define: ***SAI_MODEMASTER_RX ((uint32_t)0x00000001)***
- #define: ***SAI_MODESLAVE_TX ((uint32_t)0x00000002)***
- #define: ***SAI_MODESLAVE_RX ((uint32_t)0x00000003)***

SAI_Block_MSB_LSB_transmission

- #define: ***SAI_FIRSTBIT_MSB ((uint32_t)0x00000000)***

- #define: ***SAI_FIRSTBIT_LSB ((uint32_t)SAI_xCR1_LSBFIRST)***

SAI_Block_Mute_Value

- #define: ***SAI_ZERO_VALUE ((uint32_t)0x00000000)***

- #define: ***SAI_LAST_SENT_VALUE ((uint32_t)SAI_xCR2_MUTEVAL)***

SAI_Block_NoDivider

- #define: ***SAI_MASTERDIVIDER_ENABLED ((uint32_t)0x00000000)***

- #define: ***SAI_MASTERDIVIDER_DISABLED ((uint32_t)SAI_xCR1_NODIV)***

SAI_Block_Output_Drive

- #define: ***SAI_OUTPUTDRIVE_DISABLED ((uint32_t)0x00000000)***

- #define: ***SAI_OUTPUTDRIVE_ENABLED ((uint32_t)SAI_xCR1_OUTDRIV)***

SAI_Block_Protocol

- #define: ***SAI_FREE_PROTOCOL ((uint32_t)0x00000000)***

- #define: ***SAI_AC97_PROTOCOL ((uint32_t)SAI_xCR1_PRTCFG_1)***

SAI_Block_Slot_Active

- #define: ***SAI_SLOT_NOTACTIVE ((uint32_t)0x00000000)***

- #define: ***SAI_SLOTACTIVE_0 ((uint32_t)0x00010000)***
- #define: ***SAI_SLOTACTIVE_1 ((uint32_t)0x00020000)***
- #define: ***SAI_SLOTACTIVE_2 ((uint32_t)0x00040000)***
- #define: ***SAI_SLOTACTIVE_3 ((uint32_t)0x00080000)***
- #define: ***SAI_SLOTACTIVE_4 ((uint32_t)0x00100000)***
- #define: ***SAI_SLOTACTIVE_5 ((uint32_t)0x00200000)***
- #define: ***SAI_SLOTACTIVE_6 ((uint32_t)0x00400000)***
- #define: ***SAI_SLOTACTIVE_7 ((uint32_t)0x00800000)***
- #define: ***SAI_SLOTACTIVE_8 ((uint32_t)0x01000000)***
- #define: ***SAI_SLOTACTIVE_9 ((uint32_t)0x02000000)***
- #define: ***SAI_SLOTACTIVE_10 ((uint32_t)0x04000000)***
- #define: ***SAI_SLOTACTIVE_11 ((uint32_t)0x08000000)***

- #define: ***SAI_SLOTACTIVE_12 ((uint32_t)0x10000000)***
- #define: ***SAI_SLOTACTIVE_13 ((uint32_t)0x20000000)***
- #define: ***SAI_SLOTACTIVE_14 ((uint32_t)0x40000000)***
- #define: ***SAI_SLOTACTIVE_15 ((uint32_t)0x80000000)***
- #define: ***SAI_SLOTACTIVE_ALL ((uint32_t)0xFFFF0000)***

SAI_Block_Slot_Size

- #define: ***SAI_SLOTSIZE_DATASIZE ((uint32_t)0x00000000)***
- #define: ***SAI_SLOTSIZE_16B ((uint32_t)SAI_xSLOTR_SLOTSZ_0)***
- #define: ***SAI_SLOTSIZE_32B ((uint32_t)SAI_xSLOTR_SLOTSZ_1)***

SAI_Block_Synchronization

- #define: ***SAI_ASYNCHRONOUS ((uint32_t)0x00000000)***
- #define: ***SAI_SYNCHRONOUS ((uint32_t)SAI_xCR1_SYNCEN_0)***

SAI_Clock_Source

- #define: ***SAI_CLKSOURCE_PLLSAI ((uint32_t)RCC_SAIACLKSOURCE_PLLSAI)***

- #define: ***SAI_CLKSOURCE_PLLI2S ((uint32_t)RCC_SAIACLKSOURCE_PLLI2S)***

- #define: ***SAI_CLKSOURCE_EXT ((uint32_t)RCC_SAIACLKSOURCE_EXT)***

SAI_Mono_Stereo_Mode

- #define: ***SAI_MONOMODE ((uint32_t)SAI_xCR1_MONO)***

- #define: ***SAI_STREOMODE ((uint32_t)0x00000000)***

SAI_TRIState_Management

- #define: ***SAI_OUTPUT_NOTRELEASED ((uint32_t)0x00000000)***

- #define: ***SAI_OUTPUT_RELEASED ((uint32_t)SAI_xCR2_TRIS)***

42 HAL SMARTCARD Generic Driver

42.1 SMARTCARD Firmware driver registers structures

42.1.1 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***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef State***
- ***__IO HAL_SMARTCARD_ErrorTypeDef ErrorCode***

Field Documentation

- ***USART_TypeDef* SMARTCARD_HandleTypeDef::Instance***
- ***SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init***
- ***uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr***
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferSize***
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferCount***
- ***uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr***
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferSize***
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx***
- ***HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State***
- ***__IO HAL_SMARTCARD_ErrorTypeDef SMARTCARD_HandleTypeDef::ErrorCode***

42.1.2 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) / (8 * (hirda->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 8) + 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***
- ***uint32_t SMARTCARD_InitTypeDef::Mode***
  – Specifies wether 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. This parameter must be a number between Min_Data = 0 and Max_Data = 255
- ***uint32_t SMARTCARD_InitTypeDef::GuardTime***
  – Specifies the SmartCard Guard Time. This parameter must be a number between Min_Data = 0 and Max_Data = 255
- ***uint32_t SMARTCARD_InitTypeDef::NACKState***
  – Specifies the SmartCard NACK Transmission state. This parameter can be a value of ***SmartCard_NACK_State***

42.1.3 USART_TypeDef

USART_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t BRR***
- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t CR3***
- ***__IO uint32_t GTPR***

Field Documentation

- ***__IO uint32_t USART_TypeDef::SR***
  - USART Status register, Address offset: 0x00
- ***__IO uint32_t USART_TypeDef::DR***
  - USART Data register, Address offset: 0x04
- ***__IO uint32_t USART_TypeDef::BRR***
  - USART Baud rate register, Address offset: 0x08
- ***__IO uint32_t USART_TypeDef::CR1***
  - USART Control register 1, Address offset: 0x0C
- ***__IO uint32_t USART_TypeDef::CR2***
  - USART Control register 2, Address offset: 0x10
- ***__IO uint32_t USART_TypeDef::CR3***
  - USART Control register 3, Address offset: 0x14
- ***__IO uint32_t USART_TypeDef::GTPR***
  - USART Guard time and prescaler register, Address offset: 0x18

42.2 SMARTCARD Firmware driver API description

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

42.2.1 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 USARTx interface clock.
   b. SMARTCARD pins configuration:
      - Enable the clock for the SMARTCARD GPIOs.
      - Configure these 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 initilalized 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.

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 customed HAL_SMARTCARD_MspInit() API.

The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __SMARTCARD_ENABLE_IT() and __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

You can refer to the SMARTCARD HAL driver header file for more useful macros

42.2.2 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 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. Refer to the STM32F4xx reference manual (RM0090) for the SMARTCARD frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - 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. -@- 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 procedure (details for the procedure are available in reference manual (RM0329)).

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***

42.2.3 IO operation functions

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 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 respectivelly at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
   - HAL_SMARTCARD_Transmit()
   - HAL_SMARTCARD_Receive()
3. Non Blocking mode APIs with Interrupt are :
   - HAL_SMARTCARD_Transmit_IT()
   - HAL_SMARTCARD_Receive_IT()
   - HAL_SMARTCARD_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_SMARTCARD_Transmit_DMA()
   - HAL_SMARTCARD_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_SMARTCARD_TxCpltCallback()
   - HAL_SMARTCARD_RxCpltCallback()

- HAL_SMARTCARD_ErrorCallback()
- ***HAL_SMARTCARD_Transmit()***
- ***HAL_SMARTCARD_Receive()***
- ***HAL_SMARTCARD_Transmit_IT()***
- ***HAL_SMARTCARD_Receive_IT()***
- ***HAL_SMARTCARD_Transmit_DMA()***
- ***HAL_SMARTCARD_Receive_DMA()***
- ***HAL_SMARTCARD_IRQHandler()***
- ***HAL_SMARTCARD_TxCpltCallback()***
- ***HAL_SMARTCARD_RxCpltCallback()***
- ***HAL_SMARTCARD_ErrorCallback()***

42.2.4 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 occured durung communication.
- ***HAL_SMARTCARD_GetState()***
- ***HAL_SMARTCARD_GetError()***

42.2.5 SmartCard Initialization and de-initialization functions

42.2.5.1 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**
Notes	• None.

42.2.5.2 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**
Notes	• None.

42.2.5.3 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.
Notes	• None.

42.2.5.4 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.
Notes	• None.

42.2.6 IO operation functions

42.2.6.1 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**
Notes	• None.

42.2.6.2 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**
Notes	• None.

42.2.6.3 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**
Notes	• None.

42.2.6.4 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**
Notes	• None.

42.2.6.5 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**
Notes	• None.

42.2.6.6 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

42.2.6.7 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.
Notes	• None.

42.2.6.8 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.
Notes	• None.

42.2.6.9 HAL_SMARTCARD_RxCpltCallback

Function Name	**void HAL_SMARTCARD_RxCpltCallback (*SMARTCARD_HandleTypeDef* * hsc)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **hsc :** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
Return values	• None.
Notes	• None.

42.2.6.10 HAL_SMARTCARD_ErrorCallback

Function Name	**void HAL_SMARTCARD_ErrorCallback (*SMARTCARD_HandleTypeDef* * hsc)**
Function Description	SMARTCARD error callbacks.
Parameters	• **hsc :** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
Return values	• None.

Notes	• None.

42.2.7 Peripheral State and Errors functions

42.2.7.1 HAL_SMARTCARD_GetState

Function Name	**HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (*SMARTCARD_HandleTypeDef ** hsc)**
Function Description	return the SMARTCARD state
Parameters	• **hsc :** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.
Return values	• **HAL state**
Notes	• None.

42.2.7.2 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**
Notes	• None.

42.3 SMARTCARD Firmware driver defines

42.3.1 SMARTCARD

SMARTCARD

SMARTCARD_Clock_Phase

- #define: ***SMARTCARD_PHASE_1EDGE ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_PHASE_2EDGE ((uint32_t)USART_CR2_CPHA)***

SMARTCARD_Clock_Polarity

- #define: ***SMARTCARD_POLARITY_LOW ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_POLARITY_HIGH ((uint32_t)USART_CR2_CPOL)***

SmartCard_DMA_Requests

- #define: ***SMARTCARD_DMAREQ_TX ((uint32_t)USART_CR3_DMAT)***
- #define: ***SMARTCARD_DMAREQ_RX ((uint32_t)USART_CR3_DMAR)***

SmartCard_Flags

- #define: ***SMARTCARD_FLAG_TXE ((uint32_t)0x00000080)***
- #define: ***SMARTCARD_FLAG_TC ((uint32_t)0x00000040)***
- #define: ***SMARTCARD_FLAG_RXNE ((uint32_t)0x00000020)***
- #define: ***SMARTCARD_FLAG_IDLE ((uint32_t)0x00000010)***
- #define: ***SMARTCARD_FLAG_ORE ((uint32_t)0x00000008)***

- #define: ***SMARTCARD_FLAG_NE ((uint32_t)0x00000004)***
- #define: ***SMARTCARD_FLAG_FE ((uint32_t)0x00000002)***
- #define: ***SMARTCARD_FLAG_PE ((uint32_t)0x00000001)***

SmartCard_Interrupt_definition

- #define: ***SMARTCARD_IT_PE ((uint32_t)0x10000100)***
- #define: ***SMARTCARD_IT_TXE ((uint32_t)0x10000080)***
- #define: ***SMARTCARD_IT_TC ((uint32_t)0x10000040)***
- #define: ***SMARTCARD_IT_RXNE ((uint32_t)0x10000020)***
- #define: ***SMARTCARD_IT_IDLE ((uint32_t)0x10000010)***
- #define: ***SMARTCARD_IT_ERR ((uint32_t)0x20000001)***

SMARTCARD_Last_Bit

- #define: ***SMARTCARD_LASTBIT_DISABLE ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_LASTBIT_ENABLE ((uint32_t)USART_CR2_LBCL)***

SMARTCARD_Mode

- #define: ***SMARTCARD_MODE_RX ((uint32_t)USART_CR1_RE)***
- #define: ***SMARTCARD_MODE_TX ((uint32_t)USART_CR1_TE)***
- #define: ***SMARTCARD_MODE_TX_RX ((uint32_t)(USART_CR1_TE |USART_CR1_RE))***

SmartCard_NACK_State

- #define: ***SMARTCARD_NACK_ENABLED ((uint32_t)USART_CR3_NACK)***
- #define: ***SMARTCARD_NACK_DISABLED ((uint32_t)0x00000000)***

SMARTCARD_Parity

- #define: ***SMARTCARD_PARITY_NONE ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_PARITY_EVEN ((uint32_t)USART_CR1_PCE)***
- #define: ***SMARTCARD_PARITY_ODD ((uint32_t)(USART_CR1_PCE | USART_CR1_PS))***

SMARTCARD_Stop_Bits

- #define: ***SMARTCARD_STOPBITS_1 ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_STOPBITS_0_5 ((uint32_t)USART_CR2_STOP_0)***
- #define: ***SMARTCARD_STOPBITS_2 ((uint32_t)USART_CR2_STOP_1)***

- #define: ***SMARTCARD_STOPBITS_1_5 ((uint32_t)(USART_CR2_STOP_0 | USART_CR2_STOP_1))***

SMARTCARD_Word_Length

- #define: ***SMARTCARD_WORDLENGTH_8B ((uint32_t)0x00000000)***

- #define: ***SMARTCARD_WORDLENGTH_9B ((uint32_t)USART_CR1_M)***

43 HAL SRAM Generic Driver

43.1 SRAM Firmware driver registers structures

43.1.1 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

43.2 SRAM Firmware driver API description

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

43.2.1 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()

43.2.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

- ***HAL_SRAM_Init()***
- ***HAL_SRAM_DeInit()***
- ***HAL_SRAM_MspInit()***
- ***HAL_SRAM_MspDeInit()***
- ***HAL_SRAM_DMA_XferCpltCallback()***
- ***HAL_SRAM_DMA_XferErrorCallback()***

43.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

- ***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()***

43.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

- *HAL_SRAM_WriteOperation_Enable()*
- *HAL_SRAM_WriteOperation_Disable()*

43.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

- *HAL_SRAM_GetState()*

43.2.6 Initialization and de-initialization functions

43.2.6.1 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**
Notes	• None.

43.2.6.2 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**

Notes	• None.

43.2.6.3 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.
Notes	• None.

43.2.6.4 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.
Notes	• None.

43.2.6.5 HAL_SRAM_DMA_XferCpltCallback

Function Name	**void HAL_SRAM_DMA_XferCpltCallback (** ***DMA_HandleTypeDef **** **hdma)**
Function Description	DMA transfer complete callback.
Parameters	• **hsram :** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values	•	None.
Notes	•	None.

43.2.6.6 HAL_SRAM_DMA_XferErrorCallback

Function Name	**void HAL_SRAM_DMA_XferErrorCallback (*DMA_HandleTypeDef ** hdma)**
Function Description	DMA transfer complete error callback.
Parameters	• **hsram :** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• None.
Notes	• None.

43.2.7 Input and Output functions

43.2.7.1 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**
Notes	• None.

43.2.7.2 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**
Notes	• None.

43.2.7.3 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**
Notes	• None.

43.2.7.4 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**
Notes	• None.

43.2.7.5 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**
Notes	• None.

43.2.7.6 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**
Notes	• None.

43.2.7.7 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**
Notes	• None.

43.2.7.8 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**
Notes	• None.

43.2.8 Control functions

43.2.8.1 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**
Notes	• None.

43.2.8.2 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**
Notes	• None.

43.2.9 State functions

43.2.9.1 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**
Notes	• None.

43.3 SRAM Firmware driver defines

43.3.1 SRAM

SRAM

44 HAL SDRAM Generic Driver

44.1 SDRAM Firmware driver registers structures

44.1.1 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

44.2 SDRAM Firmware driver API description

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

44.2.1 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()

44.2.2 SDRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

- ***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()***

44.2.3 SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

- ***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()***

44.2.4 SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

- *HAL_SDRAM_WriteProtection_Enable()*
- *HAL_SDRAM_WriteProtection_Disable()*
- *HAL_SDRAM_SendCommand()*
- *HAL_SDRAM_ProgramRefreshRate()*
- *HAL_SDRAM_SetAutoRefreshNumber()*
- *HAL_SDRAM_GetModeStatus()*

44.2.5 SDRAM State functions

This subsection permits to get in run-time the status of the SDRAM controller and the data flow.

- *HAL_SDRAM_GetState()*

44.2.6 Initialization and de-initialization functions

44.2.6.1 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**
Notes	• None.

44.2.6.2 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**
Notes	• None.

44.2.6.3 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.
Notes	• None.

44.2.6.4 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.
Notes	• None.

44.2.6.5 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**
Notes	• None.

44.2.6.6 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.
Notes	• None.

44.2.6.7 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.
Notes	• None.

44.2.6.8 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.
Notes	• None.

44.2.7 Input and Output functions

44.2.7.1 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**
Notes	• None.

44.2.7.2 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**
Notes	• None.

44.2.7.3 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**
Notes	• None.

44.2.7.4 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**
Notes	• None.

44.2.7.5 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**
Notes	• None.

44.2.7.6 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**
Notes	• None.

44.2.7.7 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**
Notes	• None.

44.2.7.8 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**
Notes	• None.

44.2.8 Control functions

44.2.8.1 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**
Notes	• None.

44.2.8.2 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**
Notes	• None.

44.2.8.3 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**
Notes	• None.

44.2.8.4 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**
Notes	• None.

44.2.8.5 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**
Notes	• None.

44.2.8.6 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.**
Notes	• None.

44.2.9 State functions

44.2.9.1 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**
Notes	• None.

44.3 SDRAM Firmware driver defines

44.3.1 SDRAM

SDRAM

45 HAL SPI Generic Driver

45.1 SPI Firmware driver registers structures

45.1.1 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***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***void(* RxISR***
- ***void(* TxISR***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO HAL_SPI_ErrorTypeDef ErrorCode***

Field Documentation

- ***SPI_TypeDef* SPI_HandleTypeDef::Instance***
- ***SPI_InitTypeDef SPI_HandleTypeDef::Init***
- ***uint8_t* SPI_HandleTypeDef::pTxBuffPtr***
- ***uint16_t SPI_HandleTypeDef::TxXferSize***
- ***uint16_t SPI_HandleTypeDef::TxXferCount***
- ***uint8_t* SPI_HandleTypeDef::pRxBuffPtr***
- ***uint16_t SPI_HandleTypeDef::RxXferSize***
- ***uint16_t SPI_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* SPI_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* SPI_HandleTypeDef::hdmarx***
- ***void(* SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
- ***void(* SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
- ***HAL_LockTypeDef SPI_HandleTypeDef::Lock***
- ***__IO HAL_SPI_StateTypeDef SPI_HandleTypeDef::State***
- ***__IO HAL_SPI_ErrorTypeDef SPI_HandleTypeDef::ErrorCode***

45.1.2 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 Directional mode state. This parameter can be a value of ***SPI_Direction_mode***
- ***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***
- ***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 a number between Min_Data = 0 and Max_Data = 65535

45.1.3 SPI_TypeDef

SPI_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t CRCPR***
- ***__IO uint32_t RXCRCR***
- ***__IO uint32_t TXCRCR***
- ***__IO uint32_t I2SCFGR***
- ***__IO uint32_t I2SPR***

Field Documentation

- ***__IO uint32_t SPI_TypeDef::CR1***
  - SPI control register 1 (not used in I2S mode), Address offset: 0x00
- ***__IO uint32_t SPI_TypeDef::CR2***
  - SPI control register 2, Address offset: 0x04
- ***__IO uint32_t SPI_TypeDef::SR***
  - SPI status register, Address offset: 0x08
- ***__IO uint32_t SPI_TypeDef::DR***
  - SPI data register, Address offset: 0x0C
- ***__IO uint32_t SPI_TypeDef::CRCPR***
  - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10
- ***__IO uint32_t SPI_TypeDef::RXCRCR***
  - SPI RX CRC register (not used in I2S mode), Address offset: 0x14
- ***__IO uint32_t SPI_TypeDef::TXCRCR***
  - SPI TX CRC register (not used in I2S mode), Address offset: 0x18
- ***__IO uint32_t SPI_TypeDef::I2SCFGR***
  - SPI_I2S configuration register, Address offset: 0x1C
- ***__IO uint32_t SPI_TypeDef::I2SPR***
  - SPI_I2S prescaler register, Address offset: 0x20

45.2 SPI Firmware driver API description

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

45.2.1 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
   - Enable the DMAx interface clock using
   - Configure the DMA handle parameters
   - Configure the DMA Tx or Rx Stream
   - Associate the initilalized hdma_tx 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

3. Program the Mode, Direction , 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 customed HAL_SPI_MspInit() API.

45.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze 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 periperal.
- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***

45.2.3 IO operation functions

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 respectivelly 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. Blocking mode APIs are :
   - HAL_SPI_Transmit()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive() in full duplex mode
3. Non Blocking mode API's with Interrupt are :
   - HAL_SPI_Transmit_IT()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive_IT() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive_IT()in full duplex mode
   - HAL_SPI_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_SPI_Transmit_DMA()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceie_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_SPI_TxCpltCallback()
   - HAL_SPI_RxCpltCallback()
   - HAL_SPI_ErrorCallback()
   - HAL_SPI_TxRxCpltCallback()

- ***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_IRQHandler()***
- ***HAL_SPI_TxCpltCallback()***
- ***HAL_SPI_RxCpltCallback()***
- ***HAL_SPI_TxRxCpltCallback()***
- ***HAL_SPI_ErrorCallback()***

45.2.4 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
- ***HAL_SPI_GetState()***
- ***HAL_SPI_GetError()***

45.2.5 Initialization and de-initialization functions

45.2.5.1 HAL_SPI_Init

Function Name	**HAL_StatusTypeDef HAL_SPI_Init (*SPI_HandleTypeDef ** hspi)**
Function Description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL status**
Notes	• None.

45.2.5.2 HAL_SPI_DeInit

Function Name	**HAL_StatusTypeDef HAL_SPI_DeInit (*SPI_HandleTypeDef ** hspi)**
Function Description	DeInitializes the SPI peripheral.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL status**
Notes	• None.

45.2.5.3 HAL_SPI_MspInit

Function Name	**void HAL_SPI_MspInit (*SPI_HandleTypeDef ** hspi)**
Function Description	SPI MSP Init.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None.
Notes	• None.

45.2.5.4 HAL_SPI_MspDeInit

Function Name	**void HAL_SPI_MspDeInit (*SPI_HandleTypeDef ** hspi)**
Function Description	SPI MSP DeInit.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None.
Notes	• None.

45.2.6 IO operation functions

45.2.6.1 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**
Notes	• None.

45.2.6.2 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 sent • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

45.2.6.3 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 to be • **Size :** amount of data to be sent • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

45.2.6.4 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 no-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**

Notes	• None.

45.2.6.5 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 no-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**
Notes	• None.

45.2.6.6 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 no-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 to be • **Size :** amount of data to be sent
Return values	• **HAL status**
Notes	• None.

45.2.6.7 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 no-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	• None.

45.2.6.8 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 no-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
Parameters	• **Size :** amount of data to be sent
Return values	• **HAL status**
Notes	• When the CRC feature is enabled the pData Length must be Size + 1.

45.2.6.9 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 no-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
Parameters	• **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

45.2.6.10 HAL_SPI_IRQHandler

Function Name	**void HAL_SPI_IRQHandler (** ***SPI_HandleTypeDef **** **hspi)**
Function Description	This function handles SPI interrupt request.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL status**
Notes	• None.

45.2.6.11 HAL_SPI_TxCpltCallback

Function Name	**void HAL_SPI_TxCpltCallback (** ***SPI_HandleTypeDef **** **hspi)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None.
Notes	• None.

45.2.6.12 HAL_SPI_RxCpltCallback

Function Name	**void HAL_SPI_RxCpltCallback (** ***SPI_HandleTypeDef **** **hspi)**

Function Description	Rx Transfer completed callbacks.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None.
Notes	• None.

45.2.6.13 HAL_SPI_TxRxCpltCallback

Function Name	**void HAL_SPI_TxRxCpltCallback (** ***SPI_HandleTypeDef **** **hspi)**
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None.
Notes	• None.

45.2.6.14 HAL_SPI_ErrorCallback

Function Name	**void HAL_SPI_ErrorCallback (** ***SPI_HandleTypeDef **** **hspi)**
Function Description	SPI error callbacks.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None.
Notes	• None.

45.2.7 Peripheral State and Errors functions

45.2.7.1 HAL_SPI_GetState

Function Name	**HAL_SPI_StateTypeDef HAL_SPI_GetState (*SPI_HandleTypeDef ** hspi)**
Function Description	Return the SPI state.
Parameters	• **hspi :** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• **HAL state**
Notes	• None.

45.2.7.2 HAL_SPI_GetError

Function Name	**HAL_SPI_ErrorTypeDef 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**
Notes	• None.

45.3 SPI Firmware driver defines

45.3.1 SPI

SPI

SPI_BaudRate_Prescaler

- #define: ***SPI_BAUDRATEPRESCALER_2 ((uint32_t)0x00000000)***
- #define: ***SPI_BAUDRATEPRESCALER_4 ((uint32_t)0x00000008)***
- #define: ***SPI_BAUDRATEPRESCALER_8 ((uint32_t)0x00000010)***

- #define: ***SPI_BAUDRATEPRESCALER_16 ((uint32_t)0x00000018)***

- #define: ***SPI_BAUDRATEPRESCALER_32 ((uint32_t)0x00000020)***

- #define: ***SPI_BAUDRATEPRESCALER_64 ((uint32_t)0x00000028)***

- #define: ***SPI_BAUDRATEPRESCALER_128 ((uint32_t)0x00000030)***

- #define: ***SPI_BAUDRATEPRESCALER_256 ((uint32_t)0x00000038)***

SPI_Clock_Phase

- #define: ***SPI_PHASE_1EDGE ((uint32_t)0x00000000)***

- #define: ***SPI_PHASE_2EDGE SPI_CR1_CPHA***

SPI_Clock_Polarity

- #define: ***SPI_POLARITY_LOW ((uint32_t)0x00000000)***

- #define: ***SPI_POLARITY_HIGH SPI_CR1_CPOL***

SPI_CRC_Calculation

- #define: ***SPI_CRCCALCULATION_DISABLED ((uint32_t)0x00000000)***

- #define: ***SPI_CRCCALCULATION_ENABLED SPI_CR1_CRCEN***

SPI_data_size

- #define: ***SPI_DATASIZE_8BIT ((uint32_t)0x00000000)***
- #define: ***SPI_DATASIZE_16BIT SPI_CR1_DFF***

SPI_Direction_mode

- #define: ***SPI_DIRECTION_2LINES ((uint32_t)0x00000000)***
- #define: ***SPI_DIRECTION_2LINES_RXONLY SPI_CR1_RXONLY***
- #define: ***SPI_DIRECTION_1LINE SPI_CR1_BIDIMODE***

SPI_Flag_definition

- #define: ***SPI_FLAG_RXNE SPI_SR_RXNE***
- #define: ***SPI_FLAG_TXE SPI_SR_TXE***
- #define: ***SPI_FLAG_CRCERR SPI_SR_CRCERR***
- #define: ***SPI_FLAG_MODF SPI_SR_MODF***
- #define: ***SPI_FLAG_OVR SPI_SR_OVR***
- #define: ***SPI_FLAG_BSY SPI_SR_BSY***

- #define: ***SPI_FLAG_FRE SPI_SR_FRE***

SPI_Interrupt_configuration_definition

- #define: ***SPI_IT_TXE SPI_CR2_TXEIE***
- #define: ***SPI_IT_RXNE SPI_CR2_RXNEIE***
- #define: ***SPI_IT_ERR SPI_CR2_ERRIE***

SPI_mode

- #define: ***SPI_MODE_SLAVE ((uint32_t)0x00000000)***
- #define: ***SPI_MODE_MASTER (SPI_CR1_MSTR | SPI_CR1_SSI)***

SPI_MSB_LSB_transmission

- #define: ***SPI_FIRSTBIT_MSB ((uint32_t)0x00000000)***
- #define: ***SPI_FIRSTBIT_LSB SPI_CR1_LSBFIRST***

SPI_Slave_Select_management

- #define: ***SPI_NSS_SOFT SPI_CR1_SSM***
- #define: ***SPI_NSS_HARD_INPUT ((uint32_t)0x00000000)***
- #define: ***SPI_NSS_HARD_OUTPUT ((uint32_t)0x00040000)***

SPI_TI_mode

- #define: ***SPI_TIMODE_DISABLED ((uint32_t)0x00000000)***

- #define: ***SPI_TIMODE_ENABLED SPI_CR2_FRF***

46 HAL TIM Generic Driver

46.1 TIM Firmware driver registers structures

46.1.1 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***

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

46.1.2 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***

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 modethe number of half PWM period in center-aligned mode This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. This parameter is valid only for TIM1 and TIM8.

46.1.3 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***

- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  - Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
- ***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***
- ***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***

46.1.4 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

46.1.5 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***
- ***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***
- ***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***
- ***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

46.1.6 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

46.1.7 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 can be a value of ***TIM_ClearInput_Prescaler***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  - TIM Clear Input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

46.1.8 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

46.1.9 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_Input_Capture_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_Input_Capture_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

46.1.10 TIM_TypeDef

TIM_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t SMCR***
- ***__IO uint32_t DIER***
- ***__IO uint32_t SR***
- ***__IO uint32_t EGR***
- ***__IO uint32_t CCMR1***
- ***__IO uint32_t CCMR2***
- ***__IO uint32_t CCER***
- ***__IO uint32_t CNT***
- ***__IO uint32_t PSC***
- ***__IO uint32_t ARR***
- ***__IO uint32_t RCR***
- ***__IO uint32_t CCR1***
- ***__IO uint32_t CCR2***
- ***__IO uint32_t CCR3***
- ***__IO uint32_t CCR4***
- ***__IO uint32_t BDTR***
- ***__IO uint32_t DCR***
- ***__IO uint32_t DMAR***
- ***__IO uint32_t OR***

Field Documentation

- ***__IO uint32_t TIM_TypeDef::CR1***
  - TIM control register 1, Address offset: 0x00
- ***__IO uint32_t TIM_TypeDef::CR2***
  - TIM control register 2, Address offset: 0x04
- ***__IO uint32_t TIM_TypeDef::SMCR***
  - TIM slave mode control register, Address offset: 0x08
- ***__IO uint32_t TIM_TypeDef::DIER***

- TIM DMA/interrupt enable register, Address offset: 0x0C

- ***__IO uint32_t TIM_TypeDef::SR***
  - TIM status register, Address offset: 0x10
- ***__IO uint32_t TIM_TypeDef::EGR***
  - TIM event generation register, Address offset: 0x14
- ***__IO uint32_t TIM_TypeDef::CCMR1***
  - TIM capture/compare mode register 1, Address offset: 0x18
- ***__IO uint32_t TIM_TypeDef::CCMR2***
  - TIM capture/compare mode register 2, Address offset: 0x1C
- ***__IO uint32_t TIM_TypeDef::CCER***
  - TIM capture/compare enable register, Address offset: 0x20
- ***__IO uint32_t TIM_TypeDef::CNT***
  - TIM counter register, Address offset: 0x24
- ***__IO uint32_t TIM_TypeDef::PSC***
  - TIM prescaler, Address offset: 0x28
- ***__IO uint32_t TIM_TypeDef::ARR***
  - TIM auto-reload register, Address offset: 0x2C
- ***__IO uint32_t TIM_TypeDef::RCR***
  - TIM repetition counter register, Address offset: 0x30
- ***__IO uint32_t TIM_TypeDef::CCR1***
  - TIM capture/compare register 1, Address offset: 0x34
- ***__IO uint32_t TIM_TypeDef::CCR2***
  - TIM capture/compare register 2, Address offset: 0x38
- ***__IO uint32_t TIM_TypeDef::CCR3***
  - TIM capture/compare register 3, Address offset: 0x3C
- ***__IO uint32_t TIM_TypeDef::CCR4***
  - TIM capture/compare register 4, Address offset: 0x40
- ***__IO uint32_t TIM_TypeDef::BDTR***
  - TIM break and dead-time register, Address offset: 0x44
- ***__IO uint32_t TIM_TypeDef::DCR***
  - TIM DMA control register, Address offset: 0x48
- ***__IO uint32_t TIM_TypeDef::DMAR***
  - TIM DMA address for full transfer, Address offset: 0x4C
- ***__IO uint32_t TIM_TypeDef::OR***
  - TIM option register, Address offset: 0x50

46.2 TIM Firmware driver API description

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

46.2.1 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

46.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
   – 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 __TIMx_CLK_ENABLE();
   b. TIM pins configuration
      – Enable the clock for the TIM GPIOs using the following function: __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()

46.2.3 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.
- ***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()***

46.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- ***HAL_TIM_Base_GetState()***
- ***HAL_TIM_OC_GetState()***
- ***HAL_TIM_PWM_GetState()***
- ***HAL_TIM_IC_GetState()***
- ***HAL_TIM_OnePulse_GetState()***
- ***HAL_TIM_Encoder_GetState()***

46.2.5 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.
- ***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()*

46.2.6 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM OPWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.
- *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()*

46.2.7 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.
- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.
- *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()*

46.2.8 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.
- ***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()***

46.2.9 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.
- ***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()***

46.2.10 IRQ handler management

This section provides Timer IRQ handler function.

- ***HAL_TIM_IRQHandler()***

46.2.11 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.
- ***HAL_TIM_OC_ConfigChannel()***
- ***HAL_TIM_IC_ConfigChannel()***
- ***HAL_TIM_PWM_ConfigChannel()***
- ***HAL_TIM_OnePulse_ConfigChannel()***
- ***HAL_TIM_DMABurst_WriteStart()***
- ***HAL_TIM_DMABurst_WriteStop()***
- ***HAL_TIM_DMABurst_ReadStart()***
- ***HAL_TIM_DMABurst_ReadStop()***
- ***HAL_TIM_GenerateEvent()***
- ***HAL_TIM_ConfigOCrefClear()***
- ***HAL_TIM_ConfigClockSource()***
- ***HAL_TIM_ConfigTI1Input()***
- ***HAL_TIM_SlaveConfigSynchronization()***
- ***HAL_TIM_ReadCapturedValue()***

46.2.12 TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback
- ***HAL_TIM_PeriodElapsedCallback()***
- ***HAL_TIM_OC_DelayElapsedCallback()***
- ***HAL_TIM_IC_CaptureCallback()***
- ***HAL_TIM_PWM_PulseFinishedCallback()***
- ***HAL_TIM_TriggerCallback()***
- ***HAL_TIM_ErrorCallback()***

46.2.13 Time Base functions

46.2.13.1 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 create the associated handle.

Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.13.2 HAL_TIM_Base_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_DeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes the TIM Base peripheral.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.13.3 HAL_TIM_Base_MspInit

Function Name	**void HAL_TIM_Base_MspInit (*TIM_HandleTypeDef ** htim)**
Function Description	Initializes the TIM Base MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.13.4 HAL_TIM_Base_MspDeInit

Function Name	**void HAL_TIM_Base_MspDeInit (*TIM_HandleTypeDef ** htim)**

Function Description	DeInitializes TIM Base MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.13.5 HAL_TIM_Base_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Start (*TIM_HandleTypeDef ** htim)**
Function Description	Starts the TIM Base generation.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.13.6 HAL_TIM_Base_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Stop (*TIM_HandleTypeDef ** htim)**
Function Description	Stops the TIM Base generation.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.13.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.13.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.13.9 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **pData :** The source Buffer address. • **Length :** The length of data to be transferred from memory to peripheral.
Return values	• **HAL status**
Notes	• None.

46.2.13.10 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.14 Peripheral State functions

46.2.14.1 HAL_TIM_Base_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (*TIM_HandleTypeDef ** htim)**
Function Description	Return the TIM Base state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL state**
Notes	• None.

46.2.14.2 HAL_TIM_OC_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (*TIM_HandleTypeDef ** htim)**
Function Description	Return the TIM OC state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL state**
Notes	• None.

46.2.14.3 HAL_TIM_PWM_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (*TIM_HandleTypeDef ** htim)**
Function Description	Return the TIM PWM state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL state**
Notes	• None.

46.2.14.4 HAL_TIM_IC_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (*TIM_HandleTypeDef ** htim)**
Function Description	Return the TIM Input Capture state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL state**
Notes	• None.

46.2.14.5 HAL_TIM_OnePulse_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (*TIM_HandleTypeDef ** htim)**
Function Description	Return the TIM One Pulse Mode state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values	• **HAL state**
Notes	• None.

46.2.14.6 HAL_TIM_Encoder_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (*TIM_HandleTypeDef ** htim)**
Function Description	Return the TIM Encoder Mode state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL state**
Notes	• None.

46.2.15 Time Output Compare functions

46.2.15.1 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 create the associated handle.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.15.2 HAL_TIM_OC_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_DeInit (** ***TIM_HandleTypeDef **** **htim)**
Function Description	DeInitializes the TIM peripheral.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.15.3 HAL_TIM_OC_MspInit

Function Name	**void HAL_TIM_OC_MspInit (** ***TIM_HandleTypeDef **** **htim)**
Function Description	Initializes the TIM Output Compare MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.15.4 HAL_TIM_OC_MspDeInit

Function Name	**void HAL_TIM_OC_MspDeInit (** ***TIM_HandleTypeDef **** **htim)**
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.15.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.15.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.15.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.15.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.15.9 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.15.10 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.16 Time PWM functions

46.2.16.1 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 create the associated handle.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.16.2 HAL_TIM_PWM_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_DeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes the TIM peripheral.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.16.3 HAL_TIM_PWM_MspInit

Function Name	**void HAL_TIM_PWM_MspInit (*TIM_HandleTypeDef ** htim)**
Function Description	Initializes the TIM PWM MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.16.4 HAL_TIM_PWM_MspDeInit

Function Name	**void HAL_TIM_PWM_MspDeInit (** ***TIM_HandleTypeDef **** **htim)**
Function Description	DeInitializes TIM PWM MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.16.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.16.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.16.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.16.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**

Notes	• None.

46.2.16.9 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.16.10 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**

Notes	• None.

46.2.17 Time Input Capture functions

46.2.17.1 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 create the associated handle.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.17.2 HAL_TIM_IC_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_DeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes the TIM peripheral.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.17.3 HAL_TIM_IC_MspInit

Function Name	**void HAL_TIM_IC_MspInit (*TIM_HandleTypeDef ** htim)**

Function Description	Initializes the TIM INput Capture MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.17.4 HAL_TIM_IC_MspDeInit

Function Name	**void HAL_TIM_IC_MspDeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes TIM Input Capture MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.17.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.17.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.17.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.17.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.17.9 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 on in DMA mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.17.10 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 on in DMA mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.18 Time One Pulse functions

46.2.18.1 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 create the associated handle.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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 wil be generated.
Return values	• **HAL status**
Notes	• None.

46.2.18.2 HAL_TIM_OnePulse_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes the TIM One Pulse.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.18.3 HAL_TIM_OnePulse_MspInit

Function Name	**void HAL_TIM_OnePulse_MspInit (*TIM_HandleTypeDef ** htim)**
Function Description	Initializes the TIM One Pulse MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.18.4 HAL_TIM_OnePulse_MspDeInit

Function Name	**void HAL_TIM_OnePulse_MspDeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes TIM One Pulse MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.18.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.18.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.18.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.18.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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**
Notes	• None.

46.2.19 Time Encoder functions

46.2.19.1 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 create the associated handle.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sConfig :** TIM Encoder Interface configuration structure
Return values	• **HAL status**
Notes	• None.

46.2.19.2 HAL_TIM_Encoder_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes the TIM Encoder interface.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

46.2.19.3 HAL_TIM_Encoder_MspInit

Function Name	**void HAL_TIM_Encoder_MspInit (*TIM_HandleTypeDef ** htim)**
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.19.4 HAL_TIM_Encoder_MspDeInit

Function Name	**void HAL_TIM_Encoder_MspDeInit (*TIM_HandleTypeDef ** htim)**
Function Description	DeInitializes TIM Encoder Interface MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.19.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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
Return values	• **HAL status**
Notes	• None.

46.2.19.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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
Return values	• **HAL status**
Notes	• None.

46.2.19.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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
Return values	• **HAL status**
Notes	• None.

46.2.19.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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
Return values	• **HAL status**
Notes	• None.

46.2.19.9 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

	• **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 • **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**
Notes	• None.

46.2.19.10 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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
Return values	• **HAL status**
Notes	• None.

46.2.20 TIM IRQ handler management

46.2.20.1 HAL_TIM_IRQHandler

Function Name	**void HAL_TIM_IRQHandler (*TIM_HandleTypeDef* * htim)**
Function Description	This function handles TIM interrupts requests.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.21 Peripheral Control functions

46.2.21.1 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sConfig :** TIM Output Compare configuration structure • **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**
Notes	• None.

46.2.21.2 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sConfig :** TIM Input Capture configuration structure • **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**
Notes	• None.

46.2.21.3 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sConfig :** TIM PWM configuration structure • **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**
Notes	• None.

46.2.21.4 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sConfig :** TIM One Pulse configuration structure • **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
- **InputChannel :** 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**

Notes
- None.

46.2.21.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **BurstBaseAddress :** TIM Base address from when the DMA will starts the Data write. This parameters can be on of the following values:
  – ***TIM_DMABase_CR1 :***
  – ***TIM_DMABase_CR2 :***
  – ***TIM_DMABase_SMCR :***
  – ***TIM_DMABase_DIER :***
  – ***TIM_DMABase_SR :***
  – ***TIM_DMABase_EGR :***
  – ***TIM_DMABase_CCMR1 :***
  – ***TIM_DMABase_CCMR2 :***
  – ***TIM_DMABase_CCER :***
  – ***TIM_DMABase_CNT :***
  – ***TIM_DMABase_PSC :***
  – ***TIM_DMABase_ARR :***
  – ***TIM_DMABase_RCR :***
  – ***TIM_DMABase_CCR1 :***
  – ***TIM_DMABase_CCR2 :***
  – ***TIM_DMABase_CCR3 :***
  – ***TIM_DMABase_CCR4 :***
  – ***TIM_DMABase_BDTR :***
  – ***TIM_DMABase_DCR :***
- **BurstRequestSrc :** TIM DMA Request sources. This parameters can be on 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	• None.

46.2.21.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **BurstRequestSrc :** TIM DMA Request sources to disable
Return values	• **HAL status**
Notes	• None.

46.2.21.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **BurstBaseAddress :** TIM Base address from when the DMA will starts the Data read. This parameters can be on of the following values: – *TIM_DMABase_CR1 :*

  - *TIM_DMABase_CR2 :*
  - *TIM_DMABase_SMCR :*
  - *TIM_DMABase_DIER :*
  - *TIM_DMABase_SR :*
  - *TIM_DMABase_EGR :*
  - *TIM_DMABase_CCMR1 :*
  - *TIM_DMABase_CCMR2 :*
  - *TIM_DMABase_CCER :*
  - *TIM_DMABase_CNT :*
  - *TIM_DMABase_PSC :*
  - *TIM_DMABase_ARR :*
  - *TIM_DMABase_RCR :*
  - *TIM_DMABase_CCR1 :*
  - *TIM_DMABase_CCR2 :*
  - *TIM_DMABase_CCR3 :*
  - *TIM_DMABase_CCR4 :*
  - *TIM_DMABase_BDTR :*
  - *TIM_DMABase_DCR :*
- **BurstRequestSrc :** TIM DMA Request sources. This parameters can be on 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	• None.

46.2.21.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **BurstRequestSrc :** TIM DMA Request sources to disable.
Return values	• **HAL status**

Notes	• None.

46.2.21.9 HAL_TIM_GenerateEvent

Function Name	**HAL_StatusTypeDef HAL_TIM_GenerateEvent (*TIM_HandleTypeDef ** htim, uint32_t EventSource)**
Function Description	Generate a software event.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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	• TIM6 and TIM7 can only generate an update event. • TIM_EventSource_COM and TIM_EventSource_Break are used only with TIM1 and TIM8.

46.2.21.10 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 :** pointer to a TIM_HandleTypeDef structure that

	contains the configuration information for TIM module. • **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 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**
Notes	• None.

46.2.21.11 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sClockSourceConfig :** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	• **HAL status**
Notes	• None.

46.2.21.12 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module..

	• **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**
Notes	• None.

46.2.21.13 HAL_TIM_SlaveConfigSynchronization

Function Name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (*TIM_HandleTypeDef ** htim, *TIM_SlaveConfigTypeDef ** sSlaveConfig)**
Function Description	Configures the TIM in Slave mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.. • **sSlaveConfig :** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	• **HAL status**
Notes	• None.

46.2.21.14 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.. • **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**
Notes	• None.

46.2.22 TIM Callbacks functions

46.2.22.1 HAL_TIM_PeriodElapsedCallback

Function Name	**void HAL_TIM_PeriodElapsedCallback (** ***TIM_HandleTypeDef **** **htim)**
Function Description	Period elapsed callback in non blocking mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.22.2 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.22.3 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.22.4 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.22.5 HAL_TIM_TriggerCallback

Function Name	**void HAL_TIM_TriggerCallback (*TIM_HandleTypeDef ** htim)**
Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.2.22.6 HAL_TIM_ErrorCallback

Function Name	**void HAL_TIM_ErrorCallback (*TIM_HandleTypeDef ** htim)**
Function Description	Timer error callback in non blocking mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

46.3 TIM Firmware driver defines

46.3.1 TIM

TIM

TIM_Channel

- #define: ***TIM_CHANNEL_1 ((uint32_t)0x0000)***

- #define: ***TIM_CHANNEL_2 ((uint32_t)0x0004)***

- #define: ***TIM_CHANNEL_3 ((uint32_t)0x0008)***

- #define: ***TIM_CHANNEL_4 ((uint32_t)0x000C)***

- #define: ***TIM_CHANNEL_ALL ((uint32_t)0x0018)***

TIM_ClearInput_Polarity

- #define: ***TIM_CLEARINPUTPOLARITY_INVERTED TIM_ETRPOLARITY_INVERTED***

Polarity for ETRx pin

- #define: ***TIM_CLEARINPUTPOLARITY_NONINVERTED TIM_ETRPOLARITY_NONINVERTED***

Polarity for ETRx pin

TIM_ClearInput_Prescaler

- #define: ***TIM_CLEARINPUTPRESCALER_DIV1 TIM_ETRPRESCALER_DIV1***

No prescaler is used

- #define: ***TIM_CLEARINPUTPRESCALER_DIV2 TIM_ETRPRESCALER_DIV2***

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

- #define: ***TIM_CLEARINPUTPRESCALER_DIV4 TIM_ETRPRESCALER_DIV4***

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

- #define: ***TIM_CLEARINPUTPRESCALER_DIV8 TIM_ETRPRESCALER_DIV8***

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

TIM_ClearInput_Source

- #define: ***TIM_CLEARINPUTSOURCE_ETR ((uint32_t)0x0001)***

- #define: ***TIM_CLEARINPUTSOURCE_NONE ((uint32_t)0x0000)***

TIM_ClockDivision

- #define: ***TIM_CLOCKDIVISION_DIV1 ((uint32_t)0x0000)***

- #define: ***TIM_CLOCKDIVISION_DIV2 (TIM_CR1_CKD_0)***

- #define: ***TIM_CLOCKDIVISION_DIV4 (TIM_CR1_CKD_1)***

TIM_Clock_Polarity

- #define: ***TIM_CLOCKPOLARITY_INVERTED TIM_ETRPOLARITY_INVERTED***

Polarity for ETRx clock sources

- #define: ***TIM_CLOCKPOLARITY_NONINVERTED TIM_ETRPOLARITY_NONINVERTED***

Polarity for ETRx clock sources

- #define: ***TIM_CLOCKPOLARITY_RISING TIM_INPUTCHANNELPOLARITY_RISING***

Polarity for TIx clock sources

- #define: ***TIM_CLOCKPOLARITY_FALLING TIM_INPUTCHANNELPOLARITY_FALLING***

Polarity for TIx clock sources

- #define: ***TIM_CLOCKPOLARITY_BOTHEDGE TIM_INPUTCHANNELPOLARITY_BOTHEDGE***

Polarity for TIx clock sources

TIM_Clock_Prescaler

- #define: ***TIM_CLOCKPRESCALER_DIV1 TIM_ETRPRESCALER_DIV1***

No prescaler is used

- #define: ***TIM_CLOCKPRESCALER_DIV2 TIM_ETRPRESCALER_DIV2***

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

- #define: ***TIM_CLOCKPRESCALER_DIV4 TIM_ETRPRESCALER_DIV4***

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

- #define: ***TIM_CLOCKPRESCALER_DIV8 TIM_ETRPRESCALER_DIV8***

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

TIM_Clock_Source

- #define: ***TIM_CLOCKSOURCE_ETRMODE2 (TIM_SMCR_ETPS_1)***

- #define: ***TIM_CLOCKSOURCE_INTERNAL (TIM_SMCR_ETPS_0)***

- #define: ***TIM_CLOCKSOURCE_ITR0 ((uint32_t)0x0000)***
- #define: ***TIM_CLOCKSOURCE_ITR1 (TIM_SMCR_TS_0)***
- #define: ***TIM_CLOCKSOURCE_ITR2 (TIM_SMCR_TS_1)***
- #define: ***TIM_CLOCKSOURCE_ITR3 (TIM_SMCR_TS_0 | TIM_SMCR_TS_1)***
- #define: ***TIM_CLOCKSOURCE_TI1ED (TIM_SMCR_TS_2)***
- #define: ***TIM_CLOCKSOURCE_TI1 (TIM_SMCR_TS_0 | TIM_SMCR_TS_2)***
- #define: ***TIM_CLOCKSOURCE_TI2 (TIM_SMCR_TS_1 | TIM_SMCR_TS_2)***
- #define: ***TIM_CLOCKSOURCE_ETRMODE1 (TIM_SMCR_TS)***

TIM_Counter_Mode

- #define: ***TIM_COUNTERMODE_UP ((uint32_t)0x0000)***
- #define: ***TIM_COUNTERMODE_DOWN TIM_CR1_DIR***
- #define: ***TIM_COUNTERMODE_CENTERALIGNED1 TIM_CR1_CMS_0***
- #define: ***TIM_COUNTERMODE_CENTERALIGNED2 TIM_CR1_CMS_1***

- #define: ***TIM_COUNTERMODE_CENTERALIGNED3 TIM_CR1_CMS***

TIM_DMA_Base_address

- #define: ***TIM_DMABase_CR1 (0x00000000)***
- #define: ***TIM_DMABase_CR2 (0x00000001)***
- #define: ***TIM_DMABase_SMCR (0x00000002)***
- #define: ***TIM_DMABase_DIER (0x00000003)***
- #define: ***TIM_DMABase_SR (0x00000004)***
- #define: ***TIM_DMABase_EGR (0x00000005)***
- #define: ***TIM_DMABase_CCMR1 (0x00000006)***
- #define: ***TIM_DMABase_CCMR2 (0x00000007)***
- #define: ***TIM_DMABase_CCER (0x00000008)***
- #define: ***TIM_DMABase_CNT (0x00000009)***
- #define: ***TIM_DMABase_PSC (0x0000000A)***

- #define: ***TIM_DMABase_ARR (0x0000000B)***
- #define: ***TIM_DMABase_RCR (0x0000000C)***
- #define: ***TIM_DMABase_CCR1 (0x0000000D)***
- #define: ***TIM_DMABase_CCR2 (0x0000000E)***
- #define: ***TIM_DMABase_CCR3 (0x0000000F)***
- #define: ***TIM_DMABase_CCR4 (0x00000010)***
- #define: ***TIM_DMABase_BDTR (0x00000011)***
- #define: ***TIM_DMABase_DCR (0x00000012)***
- #define: ***TIM_DMABase_OR (0x00000013)***

TIM_DMA_Burst_Length

- #define: ***TIM_DMABurstLength_1Transfer (0x00000000)***
- #define: ***TIM_DMABurstLength_2Transfers (0x00000100)***

- #define: ***TIM_DMABurstLength_3Transfers (0x00000200)***
- #define: ***TIM_DMABurstLength_4Transfers (0x00000300)***
- #define: ***TIM_DMABurstLength_5Transfers (0x00000400)***
- #define: ***TIM_DMABurstLength_6Transfers (0x00000500)***
- #define: ***TIM_DMABurstLength_7Transfers (0x00000600)***
- #define: ***TIM_DMABurstLength_8Transfers (0x00000700)***
- #define: ***TIM_DMABurstLength_9Transfers (0x00000800)***
- #define: ***TIM_DMABurstLength_10Transfers (0x00000900)***
- #define: ***TIM_DMABurstLength_11Transfers (0x00000A00)***
- #define: ***TIM_DMABurstLength_12Transfers (0x00000B00)***
- #define: ***TIM_DMABurstLength_13Transfers (0x00000C00)***
- #define: ***TIM_DMABurstLength_14Transfers (0x00000D00)***

- #define: ***TIM_DMABurstLength_15Transfers (0x00000E00)***
- #define: ***TIM_DMABurstLength_16Transfers (0x00000F00)***
- #define: ***TIM_DMABurstLength_17Transfers (0x00001000)***
- #define: ***TIM_DMABurstLength_18Transfers (0x00001100)***

TIM_DMA_sources

- #define: ***TIM_DMA_UPDATE (TIM_DIER_UDE)***
- #define: ***TIM_DMA_CC1 (TIM_DIER_CC1DE)***
- #define: ***TIM_DMA_CC2 (TIM_DIER_CC2DE)***
- #define: ***TIM_DMA_CC3 (TIM_DIER_CC3DE)***
- #define: ***TIM_DMA_CC4 (TIM_DIER_CC4DE)***
- #define: ***TIM_DMA_COM (TIM_DIER_COMDE)***
- #define: ***TIM_DMA_TRIGGER (TIM_DIER_TDE)***

TIM_Encoder_Mode

- #define: ***TIM_ENCODERMODE_TI1 (TIM_SMCR_SMS_0)***

- #define: ***TIM_ENCODERMODE_TI2 (TIM_SMCR_SMS_1)***

- #define: ***TIM_ENCODERMODE_TI12 (TIM_SMCR_SMS_1 | TIM_SMCR_SMS_0)***

TIM_ETR_Polarity

- #define: ***TIM_ETRPOLARITY_INVERTED (TIM_SMCR_ETP)***

Polarity for ETR source

- #define: ***TIM_ETRPOLARITY_NONINVERTED ((uint32_t)0x0000)***

Polarity for ETR source

TIM_ETR_Prescaler

- #define: ***TIM_ETRPRESCALER_DIV1 ((uint32_t)0x0000)***

No prescaler is used

- #define: ***TIM_ETRPRESCALER_DIV2 (TIM_SMCR_ETPS_0)***

ETR input source is divided by 2

- #define: ***TIM_ETRPRESCALER_DIV4 (TIM_SMCR_ETPS_1)***

ETR input source is divided by 4

- #define: ***TIM_ETRPRESCALER_DIV8 (TIM_SMCR_ETPS)***

ETR input source is divided by 8

TIM_Event_Source

- #define: ***TIM_EventSource_Update TIM_EGR_UG***

- #define: ***TIM_EventSource_CC1 TIM_EGR_CC1G***

- #define: ***TIM_EventSource_CC2 TIM_EGR_CC2G***

- #define: ***TIM_EventSource_CC3 TIM_EGR_CC3G***

- #define: ***TIM_EventSource_CC4 TIM_EGR_CC4G***

- #define: ***TIM_EventSource_COM TIM_EGR_COMG***

- #define: ***TIM_EventSource_Trigger TIM_EGR_TG***

- #define: ***TIM_EventSource_Break TIM_EGR_BG***

TIM_Flag_definition

- #define: ***TIM_FLAG_UPDATE (TIM_SR_UIF)***

- #define: ***TIM_FLAG_CC1 (TIM_SR_CC1IF)***

- #define: ***TIM_FLAG_CC2 (TIM_SR_CC2IF)***

- #define: ***TIM_FLAG_CC3 (TIM_SR_CC3IF)***

- #define: ***TIM_FLAG_CC4 (TIM_SR_CC4IF)***

- #define: ***TIM_FLAG_COM (TIM_SR_COMIF)***

- #define: ***TIM_FLAG_TRIGGER (TIM_SR_TIF)***

- #define: ***TIM_FLAG_BREAK (TIM_SR_BIF)***

- #define: ***TIM_FLAG_CC1OF (TIM_SR_CC1OF)***

- #define: ***TIM_FLAG_CC2OF (TIM_SR_CC2OF)***

- #define: ***TIM_FLAG_CC3OF (TIM_SR_CC3OF)***

- #define: ***TIM_FLAG_CC4OF (TIM_SR_CC4OF)***

TIM_Input_Capture_Polarity

- #define: ***TIM_ICPOLARITY_RISING TIM_INPUTCHANNELPOLARITY_RISING***

- #define: ***TIM_ICPOLARITY_FALLING TIM_INPUTCHANNELPOLARITY_FALLING***

- #define: ***TIM_ICPOLARITY_BOTHEDGE TIM_INPUTCHANNELPOLARITY_BOTHEDGE***

TIM_Input_Capture_Prescaler

- #define: ***TIM_ICPSC_DIV1 ((uint32_t)0x0000)***

Capture performed each time an edge is detected on the capture input

- #define: ***TIM_ICPSC_DIV2 (TIM_CCMR1_IC1PSC_0)***

Capture performed once every 2 events

- #define: ***TIM_ICPSC_DIV4 (TIM_CCMR1_IC1PSC_1)***

Capture performed once every 4 events

- #define: ***TIM_ICPSC_DIV8 (TIM_CCMR1_IC1PSC)***

Capture performed once every 8 events

TIM_Input_Capture_Selection

- #define: ***TIM_ICSELECTION_DIRECTTI (TIM_CCMR1_CC1S_0)***

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

- #define: ***TIM_ICSELECTION_INDIRECTTI (TIM_CCMR1_CC1S_1)***

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

- #define: ***TIM_ICSELECTION_TRC (TIM_CCMR1_CC1S)***

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM_Input_Channel_Polarity

- #define: ***TIM_INPUTCHANNELPOLARITY_RISING ((uint32_t)0x00000000)***

Polarity for TIx source

- #define: ***TIM_INPUTCHANNELPOLARITY_FALLING (TIM_CCER_CC1P)***

Polarity for TIx source

- #define: ***TIM_INPUTCHANNELPOLARITY_BOTHEDGE (TIM_CCER_CC1P | TIM_CCER_CC1NP)***

Polarity for TIx source

TIM_Interrupt_definition

- #define: ***TIM_IT_UPDATE (TIM_DIER_UIE)***

- #define: ***TIM_IT_CC1 (TIM_DIER_CC1IE)***

- #define: ***TIM_IT_CC2 (TIM_DIER_CC2IE)***

- #define: ***TIM_IT_CC3 (TIM_DIER_CC3IE)***

- #define: ***TIM_IT_CC4 (TIM_DIER_CC4IE)***

- #define: ***TIM_IT_COM (TIM_DIER_COMIE)***

- #define: ***TIM_IT_TRIGGER (TIM_DIER_TIE)***

- #define: ***TIM_IT_BREAK (TIM_DIER_BIE)***

TIM_One_Pulse_Mode

- #define: ***TIM_OPMODE_SINGLE (TIM_CR1_OPM)***

- #define: ***TIM_OPMODE_REPETITIVE ((uint32_t)0x0000)***

TIM_Output_Compare_and_PWM_modes

- #define: ***TIM_OCMODE_TIMING ((uint32_t)0x0000)***

- #define: ***TIM_OCMODE_ACTIVE (TIM_CCMR1_OC1M_0)***

- #define: ***TIM_OCMODE_INACTIVE (TIM_CCMR1_OC1M_1)***

- #define: ***TIM_OCMODE_TOGGLE (TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1)***

- #define: ***TIM_OCMODE_PWM1 (TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2)***

- #define: ***TIM_OCMODE_PWM2 (TIM_CCMR1_OC1M)***
- #define: ***TIM_OCMODE_FORCED_ACTIVE (TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_2)***
- #define: ***TIM_OCMODE_FORCED_INACTIVE (TIM_CCMR1_OC1M_2)***

TIM_Output_Compare_Idle_State

- #define: ***TIM_OCIDLESTATE_SET (TIM_CR2_OIS1)***
- #define: ***TIM_OCIDLESTATE_RESET ((uint32_t)0x0000)***

TIM_Output_Compare_N_Idle_State

- #define: ***TIM_OCNIDLESTATE_SET (TIM_CR2_OIS1N)***
- #define: ***TIM_OCNIDLESTATE_RESET ((uint32_t)0x0000)***

TIM_Output_Compare_N_Polarity

- #define: ***TIM_OCNPOLARITY_HIGH ((uint32_t)0x0000)***
- #define: ***TIM_OCNPOLARITY_LOW (TIM_CCER_CC1NP)***

TIM_Output_Compare_N_State

- #define: ***TIM_OUTPUTNSTATE_DISABLE ((uint32_t)0x0000)***
- #define: ***TIM_OUTPUTNSTATE_ENABLE (TIM_CCER_CC1NE)***

TIM_Output_Compare_Polarity

- #define: ***TIM_OCPOLARITY_HIGH ((uint32_t)0x0000)***
- #define: ***TIM_OCPOLARITY_LOW (TIM_CCER_CC1P)***

TIM_Output_Compare_State

- #define: ***TIM_OUTPUTSTATE_DISABLE ((uint32_t)0x0000)***
- #define: ***TIM_OUTPUTSTATE_ENABLE (TIM_CCER_CC1E)***

TIM_Output_Fast_State

- #define: ***TIM_OCFAST_DISABLE ((uint32_t)0x0000)***
- #define: ***TIM_OCFAST_ENABLE (TIM_CCMR1_OC1FE)***

TIM_Slave_Mode

- #define: ***TIM_SLAVEMODE_DISABLE ((uint32_t)0x0000)***
- #define: ***TIM_SLAVEMODE_RESET ((uint32_t)0x0004)***
- #define: ***TIM_SLAVEMODE_GATED ((uint32_t)0x0005)***
- #define: ***TIM_SLAVEMODE_TRIGGER ((uint32_t)0x0006)***

- #define: ***TIM_SLAVEMODE_EXTERNAL1 ((uint32_t)0x0007)***

TIM_TI1_Selection

- #define: ***TIM_TI1SELECTION_CH1 ((uint32_t)0x0000)***

- #define: ***TIM_TI1SELECTION_XORCOMBINATION (TIM_CR2_TI1S)***

TIM_Trigger_Polarity

- #define: ***TIM_TRIGGERPOLARITY_INVERTED TIM_ETRPOLARITY_INVERTED***

Polarity for ETRx trigger sources

- #define: ***TIM_TRIGGERPOLARITY_NONINVERTED TIM_ETRPOLARITY_NONINVERTED***

Polarity for ETRx trigger sources

- #define: ***TIM_TRIGGERPOLARITY_RISING TIM_INPUTCHANNELPOLARITY_RISING***

Polarity for TIxFPx or TI1_ED trigger sources

- #define: ***TIM_TRIGGERPOLARITY_FALLING TIM_INPUTCHANNELPOLARITY_FALLING***

Polarity for TIxFPx or TI1_ED trigger sources

- #define: ***TIM_TRIGGERPOLARITY_BOTHEDGE TIM_INPUTCHANNELPOLARITY_BOTHEDGE***

Polarity for TIxFPx or TI1_ED trigger sources

TIM_Trigger_Prescaler

- #define: ***TIM_TRIGGERPRESCALER_DIV1 TIM_ETRPRESCALER_DIV1***

No prescaler is used

- #define: ***TIM_TRIGGERPRESCALER_DIV2 TIM_ETRPRESCALER_DIV2***

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

- #define: ***TIM_TRIGGERPRESCALER_DIV4 TIM_ETRPRESCALER_DIV4***

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

- #define: ***TIM_TRIGGERPRESCALER_DIV8 TIM_ETRPRESCALER_DIV8***

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

TIM_Trigger_Selection

- #define: ***TIM_TS_ITR0 ((uint32_t)0x0000)***
- #define: ***TIM_TS_ITR1 ((uint32_t)0x0010)***
- #define: ***TIM_TS_ITR2 ((uint32_t)0x0020)***
- #define: ***TIM_TS_ITR3 ((uint32_t)0x0030)***
- #define: ***TIM_TS_TI1F_ED ((uint32_t)0x0040)***
- #define: ***TIM_TS_TI1FP1 ((uint32_t)0x0050)***
- #define: ***TIM_TS_TI2FP2 ((uint32_t)0x0060)***
- #define: ***TIM_TS_ETRF ((uint32_t)0x0070)***
- #define: ***TIM_TS_NONE ((uint32_t)0xFFFF)***

47 HAL TIM Extension Driver

47.1 TIMEx Firmware driver registers structures

47.1.1 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32f4xx_hal_tim_ex.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 ***TIMEx_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  - Master/slave mode selection. This parameter can be a value of ***TIMEx_Master_Slave_Mode***

47.1.2 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

47.1.3 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32f4xx_hal_tim_ex.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***
- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  – TIM off state in run mode. This parameter can be a value of ***TIMEx_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 ***TIMEx_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  – TIM Lock level. This parameter can be a value of ***TIMEx_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 ***TIMEx_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  – TIM Break input polarity. This parameter can be a value of ***TIMEx_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  – TIM Automatic Output Enable state. This parameter can be a value of ***TIMEx_AOE_Bit_Set_Reset***

47.2 TIMEx Firmware driver API description

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

47.2.1 TIMER Extended features

The Timer Extension features include:

1. Complementary outputs with programmable dead-time for :
   - Input Capture
   - 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

47.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
   - Complementary Output Compare : HAL_TIM_OC_MspInit()
   - Complementary PWM generation : HAL_TIM_PWM_MspInit()
   - Complementary One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   - Hall Sensor output : HAL_TIM_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __TIMx_CLK_ENABLE();
   b. TIM pins configuration
      - Enable the clock for the TIM GPIOs using the following function: __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_ConfigCommutationEvent: 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_OC_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().

47.2.3 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.
- ***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()***

47.2.4 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.
- ***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()***

47.2.5 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.
- *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()*

47.2.6 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.
- *HAL_TIMEx_OnePulseN_Start()*
- *HAL_TIMEx_OnePulseN_Stop()*
- *HAL_TIMEx_OnePulseN_Start_IT()*
- *HAL_TIMEx_OnePulseN_Stop_IT()*

47.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the commutation event in case of use of the Hall sensor interface.
- Configure the DMA Burst Mode.
- *HAL_TIMEx_ConfigCommutationEvent()*
- *HAL_TIMEx_ConfigCommutationEvent_IT()*
- *HAL_TIMEx_ConfigCommutationEvent_DMA()*
- *HAL_TIMEx_MasterConfigSynchronization()*
- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_RemapConfig()*

47.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback
- *HAL_TIMEx_CommutationCallback()*
- *HAL_TIMEx_BreakCallback()*

47.2.9 Extension Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_TIMEx_HallSensor_GetState()*

47.2.10 Timer Hall Sensor functions

47.2.10.1 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 create the associated handle.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sConfig :** TIM Hall Sensor configuration structure
Return values	• **HAL status**
Notes	• None.

47.2.10.2 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

47.2.10.3 HAL_TIMEx_HallSensor_MspInit

Function Name	**void HAL_TIMEx_HallSensor_MspInit (*TIM_HandleTypeDef ** htim)**
Function Description	Initializes the TIM Hall Sensor MSP.

Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

47.2.10.4 HAL_TIMEx_HallSensor_MspDeInit

Function Name	**void HAL_TIMEx_HallSensor_MspDeInit (** ***TIM_HandleTypeDef **** **htim)**
Function Description	DeInitializes TIM Hall Sensor MSP.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

47.2.10.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

47.2.10.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

47.2.10.7 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

47.2.10.8 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

47.2.10.9 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **pData :** The destination Buffer address. • **Length :** The length of data to be transferred from TIM peripheral to memory.
Return values	• **HAL status**
Notes	• None.

47.2.10.10 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL status**
Notes	• None.

47.2.11 Timer Complementary Output Compare functions

47.2.11.1 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1/

	TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.11.2 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.11.3 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.11.4 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.11.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4 • **pData :** The source Buffer address. • **Length :** The length of data to be transferred from memory to TIM peripheral
Return values	• **HAL status**
Notes	• None.

47.2.11.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.12 Timer Complementary PWM functions

47.2.12.1 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.12.2 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can

	be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.12.3 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.12.4 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.12.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4 • **pData :** The source Buffer address. • **Length :** The length of data to be transferred from memory to TIM peripheral
Return values	• **HAL status**
Notes	• None.

47.2.12.6 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **Channel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	• **HAL status**
Notes	• None.

47.2.13 Timer Complementary One Pulse functions

47.2.13.1 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 complemetary output.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **OutputChannel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1 / IM_CHANNEL_2
Return values	• **HAL status**
Notes	• None.

47.2.13.2 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **OutputChannel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1 / TIM_CHANNEL_2
Return values	• **HAL status**
Notes	• None.

47.2.13.3 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **OutputChannel :** TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1 / IM_CHANNEL_2
Return values	• **HAL status**
Notes	• None.

47.2.13.4 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **OutputChannel :** TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1 / IM_CHANNEL_2
Return values	• **HAL status**
Notes	• None.

47.2.14 Peripheral Control functions

47.2.14.1 HAL_TIMEx_ConfigCommutationEvent

Function Name	**HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (** *TIM_HandleTypeDef* * **htim, uint32_t InputTrigger, uint32_t CommutationSource)**
Function Description	Configure the TIM commutation event sequence.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

	• **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 / TIM_TS_ITR1 / TIM_TS_ITR2 / TIM_TS_ITR3 / TIM_TS_NONE • **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.

47.2.14.2 HAL_TIMEx_ConfigCommutationEvent_IT

Function Name	**HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (** ***TIM_HandleTypeDef **** **htim, uint32_t InputTrigger, uint32_t CommutationSource)**
Function Description	Configure the TIM commutation event sequence with interrupt.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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 / TIM_TS_ITR1 / TIM_TS_ITR2 / TIM_TS_ITR3 / TIM_TS_NONE • **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.

47.2.14.3 HAL_TIMEx_ConfigCommutationEvent_DMA

Function Name	**HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (** ***TIM_HandleTypeDef **** **htim, uint32_t InputTrigger, uint32_t CommutationSource)**
Function Description	Configure the TIM commutation event sequence with DMA.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **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 / TIM_TS_ITR1 / TIM_TS_ITR2 / TIM_TS_ITR3 / TIM_TS_NONE • **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

47.2.14.4 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sMasterConfig :** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	• **HAL status**
Notes	• None.

47.2.14.5 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 :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • **sBreakDeadTimeConfig :** pointer to a TIM_ConfigBreakDeadConfig_TypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.
Return values	• **HAL status**
Notes	• None.

47.2.14.6 HAL_TIMEx_RemapConfig

Function Name	**HAL_StatusTypeDef HAL_TIMEx_RemapConfig (** *TIM_HandleTypeDef* * **htim, uint32_t Remap)**
Function Description	Configures the TIM2, TIM5 and TIM11 Remapping input capabilities.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module..

	• **TIM_Remap :** specifies the TIM input remapping source. This parameter can be one of the following values: – *TIM_TIM2_TIM8_TRGO :* TIM2 ITR1 input is connected to TIM8 Trigger output(default) – *TIM_TIM2_ETH_PTP :* TIM2 ITR1 input is connected to ETH PTP trogger output. – *TIM_TIM2_USBFS_SOF :* TIM2 ITR1 input is connected to USB FS SOF. – *TIM_TIM2_USBHS_SOF :* TIM2 ITR1 input is connected to USB HS SOF. – *TIM_TIM5_GPIO :* TIM5 CH4 input is connected to dedicated Timer pin(default) – *TIM_TIM5_LSI :* TIM5 CH4 input is connected to LSI clock. – *TIM_TIM5_LSE :* TIM5 CH4 input is connected to LSE clock. – *TIM_TIM5_RTC :* TIM5 CH4 input is connected to RTC Output event. – *TIM_TIM11_GPIO :* TIM11 CH4 input is connected to dedicated Timer pin(default) – *TIM_TIM11_HSE :* TIM11 CH4 input is connected to HSE_RTC clock (HSE divided by a programmable prescaler)
Return values	• **HAL status**
Notes	• None.

47.2.15 Extension Callbacks functions

47.2.15.1 HAL_TIMEx_CommutationCallback

Function Name	**void HAL_TIMEx_CommutationCallback (*TIM_HandleTypeDef* * htim)**
Function Description	Hall commutation changed callback in non blocking mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

47.2.15.2 HAL_TIMEx_BreakCallback

Function Name	**void HAL_TIMEx_BreakCallback (** ***TIM_HandleTypeDef **** **htim)**
Function Description	Hall Break detection callback in non blocking mode.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• None.
Notes	• None.

47.2.16 Extension Peripheral State functions

47.2.16.1 HAL_TIMEx_HallSensor_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (** ***TIM_HandleTypeDef **** **htim)**
Function Description	Return the TIM Hall Sensor interface state.
Parameters	• **htim :** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	• **HAL state**
Notes	• None.

47.3 TIMEx Firmware driver defines

47.3.1 TIMEx

TIMEx

TIMEx_AOE_Bit_Set_Reset

- #define: ***TIM_AUTOMATICOUTPUT_ENABLE (TIM_BDTR_AOE)***

- #define: ***TIM_AUTOMATICOUTPUT_DISABLE ((uint32_t)0x0000)***

TIMEx_Break_Input_enable_disable

- #define: ***TIM_BREAK_ENABLE (TIM_BDTR_BKE)***
- #define: ***TIM_BREAK_DISABLE ((uint32_t)0x0000)***

TIMEx_Break_Polarity

- #define: ***TIM_BREAKPOLARITY_LOW ((uint32_t)0x0000)***
- #define: ***TIM_BREAKPOLARITY_HIGH (TIM_BDTR_BKP)***

TIMEx_Commutation_Mode

- #define: ***TIM_COMMUTATION_TRGI (TIM_CR2_CCUS)***
- #define: ***TIM_COMMUTATION_SOFTWARE ((uint32_t)0x0000)***

TIMEx_Lock_level

- #define: ***TIM_LOCKLEVEL_OFF ((uint32_t)0x0000)***
- #define: ***TIM_LOCKLEVEL_1 (TIM_BDTR_LOCK_0)***
- #define: ***TIM_LOCKLEVEL_2 (TIM_BDTR_LOCK_1)***
- #define: ***TIM_LOCKLEVEL_3 (TIM_BDTR_LOCK)***

TIMEx_Master_Mode_Selection

- #define: ***TIM_TRGO_RESET ((uint32_t)0x0000)***

- #define: ***TIM_TRGO_ENABLE (TIM_CR2_MMS_0)***
- #define: ***TIM_TRGO_UPDATE (TIM_CR2_MMS_1)***
- #define: ***TIM_TRGO_OC1 ((TIM_CR2_MMS_1 | TIM_CR2_MMS_0))***
- #define: ***TIM_TRGO_OC1REF (TIM_CR2_MMS_2)***
- #define: ***TIM_TRGO_OC2REF ((TIM_CR2_MMS_2 | TIM_CR2_MMS_0))***
- #define: ***TIM_TRGO_OC3REF ((TIM_CR2_MMS_2 | TIM_CR2_MMS_1))***
- #define: ***TIM_TRGO_OC4REF ((TIM_CR2_MMS_2 | TIM_CR2_MMS_1 | TIM_CR2_MMS_0))***

TIMEx_Master_Slave_Mode

- #define: ***TIM_MASTERSLAVEMODE_ENABLE ((uint32_t)0x0080)***
- #define: ***TIM_MASTERSLAVEMODE_DISABLE ((uint32_t)0x0000)***

TIMEx_OSSI_Off_State_Selection_for_Idle_mode_state

- #define: ***TIM_OSSI_ENABLE (TIM_BDTR_OSSI)***
- #define: ***TIM_OSSI_DISABLE ((uint32_t)0x0000)***

TIMEx_OSSR_Off_State_Selection_for_Run_mode_state

- #define: ***TIM_OSSR_ENABLE (TIM_BDTR_OSSR)***
- #define: ***TIM_OSSR_DISABLE ((uint32_t)0x0000)***

TIMEx_Remap

- #define: ***TIM_TIM2_TIM8_TRGO (0x00000000)***
- #define: ***TIM_TIM2_ETH_PTP (0x00000400)***
- #define: ***TIM_TIM2_USBFS_SOF (0x00000800)***
- #define: ***TIM_TIM2_USBHS_SOF (0x00000C00)***
- #define: ***TIM_TIM5_GPIO (0x00000000)***
- #define: ***TIM_TIM5_LSI (0x00000040)***
- #define: ***TIM_TIM5_LSE (0x00000080)***
- #define: ***TIM_TIM5_RTC (0x000000C0)***
- #define: ***TIM_TIM11_GPIO (0x00000000)***

- #define: ***TIM_TIM11_HSE (0x00000002)***

48 HAL UART Generic Driver

48.1 UART Firmware driver registers structures

48.1.1 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***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef State***
- ***__IO HAL_UART_ErrorTypeDef ErrorCode***

Field Documentation

- ***USART_TypeDef* UART_HandleTypeDef::Instance***
- ***UART_InitTypeDef UART_HandleTypeDef::Init***
- ***uint8_t* UART_HandleTypeDef::pTxBuffPtr***
- ***uint16_t UART_HandleTypeDef::TxXferSize***
- ***uint16_t UART_HandleTypeDef::TxXferCount***
- ***uint8_t* UART_HandleTypeDef::pRxBuffPtr***
- ***uint16_t UART_HandleTypeDef::RxXferSize***
- ***uint16_t UART_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx***
- ***HAL_LockTypeDef UART_HandleTypeDef::Lock***
- ***__IO HAL_UART_StateTypeDef UART_HandleTypeDef::State***
- ***__IO HAL_UART_ErrorTypeDef UART_HandleTypeDef::ErrorCode***

48.1.2 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***
- ***uint32_t UART_InitTypeDef::Mode***
  - Specifies wether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  - Specifies wether 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 wether 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***

48.1.3 USART_TypeDef

USART_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t BRR***
- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t CR3***
- ***__IO uint32_t GTPR***

Field Documentation

- ***__IO uint32_t USART_TypeDef::SR***
  - USART Status register, Address offset: 0x00
- ***__IO uint32_t USART_TypeDef::DR***
  - USART Data register, Address offset: 0x04
- ***__IO uint32_t USART_TypeDef::BRR***
  - USART Baud rate register, Address offset: 0x08
- ***__IO uint32_t USART_TypeDef::CR1***
  - USART Control register 1, Address offset: 0x0C
- ***__IO uint32_t USART_TypeDef::CR2***
  - USART Control register 2, Address offset: 0x10
- ***__IO uint32_t USART_TypeDef::CR3***
  - USART Control register 3, Address offset: 0x14
- ***__IO uint32_t USART_TypeDef::GTPR***
  - USART Guard time and prescaler register, Address offset: 0x18

48.2 UART Firmware driver API description

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

48.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure.
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 as alternate function pull-up.
   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.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the 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.

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.

These APIs (HAL_UART_Init() and HAL_HalfDuplex_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_UART_MspInit() API.

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 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 using HAL_UART_Receive_IT()
- 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

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()

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

You can refer to the UART HAL driver header file for more useful macros

48.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode 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. Refer to the STM32F4xx reference manual (RM0090) for the UART frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Methode

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 (RM0329)).

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***
- ***HAL_UART_DeInit()***
- ***HAL_UART_MspInit()***

- ***HAL_UART_MspDeInit()***

48.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the UART asynchronous and Half duplex data transfers.

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 APIs 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_UART_TxCpltCallback(), HAL_UART_RxCpltCallback() user callbacks will be executed respectivelly at the end of the transmit or receive process. The HAL_UART_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are:
   - HAL_UART_Transmit()
   - HAL_UART_Receive()
3. Non Blocking mode APIs with Interrupt are:
   - HAL_UART_Transmit_IT()
   - HAL_UART_Receive_IT()
   - HAL_UART_IRQHandler()
4. Non Blocking mode functions with DMA are:
   - HAL_UART_Transmit_DMA()
   - HAL_UART_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non blocking mode:
   - HAL_UART_TxCpltCallback()
   - HAL_UART_RxCpltCallback()
   - HAL_UART_ErrorCallback()

In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state HAL_UART_STATE_BUSY_TX_RX can't be useful.

- ***HAL_UART_Transmit()***
- ***HAL_UART_Receive()***
- ***HAL_UART_Transmit_IT()***
- ***HAL_UART_Receive_IT()***
- ***HAL_UART_Transmit_DMA()***
- ***HAL_UART_Receive_DMA()***
- ***HAL_UART_DMAPause()***
- ***HAL_UART_DMAResume()***
- ***HAL_UART_DMAStop()***
- ***HAL_UART_IRQHandler()***
- ***HAL_UART_TxCpltCallback()***
- ***HAL_UART_TxHalfCpltCallback()***
- ***HAL_UART_RxCpltCallback()***
- ***HAL_UART_RxHalfCpltCallback()***
- ***HAL_UART_ErrorCallback()***

48.2.4 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_LIN_SendBreak()***
- ***HAL_MultiProcessor_EnterMuteMode()***
- ***HAL_MultiProcessor_ExitMuteMode()***
- ***HAL_HalfDuplex_EnableTransmitter()***
- ***HAL_HalfDuplex_EnableReceiver()***

48.2.5 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.
- ***HAL_UART_GetState()***
- ***HAL_UART_GetError()***

48.2.6 Initialization and de-initialization functions

48.2.6.1 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**
Notes	• None.

48.2.6.2 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**
Notes	• None.

48.2.6.3 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**
Notes	• None.

48.2.6.4 HAL_MultiProcessor_Init

Function Name	**HAL_StatusTypeDef HAL_MultiProcessor_Init (** ***UART_HandleTypeDef **** **huart, uint8_t Address, uint32_t WakeUpMethode)**

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 • **WakeUpMethode :** specifies the USART wakeup method. This parameter can be one of the following values: – *UART_WAKEUPMETHODE_IDLELINE :* Wakeup by an idle line detection – *UART_WAKEUPMETHODE_ADDRESSMARK :* Wakeup by an address mark
Return values	• **HAL status**
Notes	• None.

48.2.6.5 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**
Notes	• None.

48.2.6.6 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.

Notes	• None.

48.2.6.7 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.
Notes	• None.

48.2.7 IO operation functions

48.2.7.1 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 • **Size :** Amount of data to be sent • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

48.2.7.2 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 • **Size :** Amount of data to be received • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

48.2.7.3 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 • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

48.2.7.4 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 • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• None.

48.2.7.5 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 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 • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

48.2.7.6 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 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 • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• When the UART parity is enabled (PCE = 1) the data received contain the parity bit.

48.2.7.7 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**
Notes	• None.

48.2.7.8 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**
Notes	• None.

48.2.7.9 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**

Notes	• None.

48.2.7.10 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.
Notes	• None.

48.2.7.11 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.
Notes	• None.

48.2.7.12 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.
Notes	•	None.

48.2.7.13 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.
Notes	• None.

48.2.7.14 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.
Notes	• None.

48.2.7.15 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.
Notes	• None.

48.2.8 Peripheral Control functions

48.2.8.1 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**
Notes	• None.

48.2.8.2 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**
Notes	• None.

48.2.8.3 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**
Notes	• None.

48.2.8.4 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**
Notes	• None.

48.2.8.5 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**
Notes	• None.

48.2.9 Peripheral State and Errors functions

48.2.9.1 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**
Notes	• None.

48.2.9.2 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**
Notes	• None.

48.3 UART Firmware driver defines

48.3.1 UART

UART

UART_Flags

- #define: ***UART_FLAG_CTS ((uint32_t)0x00000200)***
- #define: ***UART_FLAG_LBD ((uint32_t)0x00000100)***
- #define: ***UART_FLAG_TXE ((uint32_t)0x00000080)***
- #define: ***UART_FLAG_TC ((uint32_t)0x00000040)***
- #define: ***UART_FLAG_RXNE ((uint32_t)0x00000020)***
- #define: ***UART_FLAG_IDLE ((uint32_t)0x00000010)***
- #define: ***UART_FLAG_ORE ((uint32_t)0x00000008)***
- #define: ***UART_FLAG_NE ((uint32_t)0x00000004)***
- #define: ***UART_FLAG_FE ((uint32_t)0x00000002)***
- #define: ***UART_FLAG_PE ((uint32_t)0x00000001)***

UART_Hardware_Flow_Control

- #define: ***UART_HWCONTROL_NONE ((uint32_t)0x00000000)***

- #define: ***UART_HWCONTROL_RTS ((uint32_t)USART_CR3_RTSE)***
- #define: ***UART_HWCONTROL_CTS ((uint32_t)USART_CR3_CTSE)***
- #define: ***UART_HWCONTROL_RTS_CTS ((uint32_t)(USART_CR3_RTSE | USART_CR3_CTSE))***

UART_Interrupt_definition

- #define: ***UART_IT_PE ((uint32_t)0x10000100)***
- #define: ***UART_IT_TXE ((uint32_t)0x10000080)***
- #define: ***UART_IT_TC ((uint32_t)0x10000040)***
- #define: ***UART_IT_RXNE ((uint32_t)0x10000020)***
- #define: ***UART_IT_IDLE ((uint32_t)0x10000010)***
- #define: ***UART_IT_LBD ((uint32_t)0x20000040)***
- #define: ***UART_IT_CTS ((uint32_t)0x30000400)***
- #define: ***UART_IT_ERR ((uint32_t)0x30000001)***

UART_LIN_Break_Detection_Length

- #define: ***UART_LINBREAKDETECTLENGTH_10B ((uint32_t)0x00000000)***
- #define: ***UART_LINBREAKDETECTLENGTH_11B ((uint32_t)0x00000020)***

UART_Mode

- #define: ***UART_MODE_RX ((uint32_t)USART_CR1_RE)***
- #define: ***UART_MODE_TX ((uint32_t)USART_CR1_TE)***
- #define: ***UART_MODE_TX_RX ((uint32_t)(USART_CR1_TE |USART_CR1_RE))***

UART_Over_Sampling

- #define: ***UART_OVERSAMPLING_16 ((uint32_t)0x00000000)***
- #define: ***UART_OVERSAMPLING_8 ((uint32_t)USART_CR1_OVER8)***

UART_Parity

- #define: ***UART_PARITY_NONE ((uint32_t)0x00000000)***
- #define: ***UART_PARITY_EVEN ((uint32_t)USART_CR1_PCE)***
- #define: ***UART_PARITY_ODD ((uint32_t)(USART_CR1_PCE | USART_CR1_PS))***

UART_State

- #define: ***UART_STATE_DISABLE ((uint32_t)0x00000000)***

- #define: ***UART_STATE_ENABLE ((uint32_t)USART_CR1_UE)***

UART_Stop_Bits

- #define: ***UART_STOPBITS_1 ((uint32_t)0x00000000)***

- #define: ***UART_STOPBITS_2 ((uint32_t)USART_CR2_STOP_1)***

UART_WakeUp_functions

- #define: ***UART_WAKEUPMETHODE_IDLELINE ((uint32_t)0x00000000)***

- #define: ***UART_WAKEUPMETHODE_ADDRESSMARK ((uint32_t)0x00000800)***

UART_Word_Length

- #define: ***UART_WORDLENGTH_8B ((uint32_t)0x00000000)***

- #define: ***UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)***

49 HAL USART Generic Driver

49.1 USART Firmware driver registers structures

49.1.1 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 HAL_USART_ErrorTypeDef ErrorCode***

Field Documentation

- ***USART_TypeDef* USART_HandleTypeDef::Instance***
- ***USART_InitTypeDef USART_HandleTypeDef::Init***
- ***uint8_t* USART_HandleTypeDef::pTxBuffPtr***
- ***uint16_t USART_HandleTypeDef::TxXferSize***
- ***__IO uint16_t USART_HandleTypeDef::TxXferCount***
- ***uint8_t* USART_HandleTypeDef::pRxBuffPtr***
- ***uint16_t USART_HandleTypeDef::RxXferSize***
- ***__IO uint16_t USART_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx***
- ***HAL_LockTypeDef USART_HandleTypeDef::Lock***
- ***__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State***
- ***__IO HAL_USART_ErrorTypeDef USART_HandleTypeDef::ErrorCode***

49.1.2 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***
- ***uint32_t USART_InitTypeDef::Mode***
  - Specifies wether 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***

49.1.3 USART_TypeDef

USART_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t SR***
- ***__IO uint32_t DR***
- ***__IO uint32_t BRR***
- ***__IO uint32_t CR1***
- ***__IO uint32_t CR2***
- ***__IO uint32_t CR3***
- ***__IO uint32_t GTPR***

Field Documentation

- ***__IO uint32_t USART_TypeDef::SR***
  - USART Status register, Address offset: 0x00
- ***__IO uint32_t USART_TypeDef::DR***
  - USART Data register, Address offset: 0x04
- ***__IO uint32_t USART_TypeDef::BRR***
  - USART Baud rate register, Address offset: 0x08
- ***__IO uint32_t USART_TypeDef::CR1***
  - USART Control register 1, Address offset: 0x0C
- ***__IO uint32_t USART_TypeDef::CR2***
  - USART Control register 2, Address offset: 0x10
- ***__IO uint32_t USART_TypeDef::CR3***
  - USART Control register 3, Address offset: 0x14
- ***__IO uint32_t USART_TypeDef::GTPR***
  - USART Guard time and prescaler register, Address offset: 0x18

49.2 USART Firmware driver API description

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

49.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure.
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 these 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_IT() and HAL_USART_TransmitReceive_IT() 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 initilalized 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.

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 customed HAL_USART_MspInit(&husart) API. The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __USART_ENABLE_IT() and __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 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 using HAL_USART_Receive_IT()
- 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

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

You can refer to the USART HAL driver header file for more useful macros

49.2.2 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. Refer to the STM32F4xx reference manual (RM0090) for the USART frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual (RM0329)).

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***

49.2.3 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 respectivelly 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_TransmitReceie_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_USART_TxCpltCallback()
   - HAL_USART_RxCpltCallback()
   - HAL_USART_ErrorCallback()
   - HAL_USART_TxRxCpltCallback()

- ***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_IRQHandler()***
- ***HAL_USART_TxCpltCallback()***
- ***HAL_USART_TxHalfCpltCallback()***
- ***HAL_USART_RxCpltCallback()***
- ***HAL_USART_RxHalfCpltCallback()***
- ***HAL_USART_TxRxCpltCallback()***
- ***HAL_USART_ErrorCallback()***

49.2.4 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.
- ***HAL_USART_GetState()***
- ***HAL_USART_GetError()***

49.2.5 USART Initialization and de-initialization functions

49.2.5.1 HAL_USART_Init

Function Name	**HAL_StatusTypeDef HAL_USART_Init (** ***USART_HandleTypeDef **** **husart)**
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create 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**
Notes	• None.

49.2.5.2 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**
Notes	• None.

49.2.5.3 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.
Notes	• None.

49.2.5.4 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.
Notes	• None.

49.2.6 IO operation functions

49.2.6.1 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

- **Size :** Amount of data to be sent
- **Timeout :** Timeout duration

Return values	• **HAL status**
Notes	• None.

49.2.6.2 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 • **Size :** Amount of data to be received • **Timeout :** Timeout duration
Return values	• **HAL status**
Notes	• None.

49.2.6.3 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 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 data transmitted buffer • **pRxData :** Pointer to data received buffer • **Size :** Amount of data to be sent • **Timeout :** Timeout duration
Return values	• **HAL status**

Notes	• None.

49.2.6.4 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 • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• The USART errors are not managed to avoid the overrun error.

49.2.6.5 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 • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• None.

49.2.6.6 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 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 data transmitted buffer • **pRxData :** Pointer to data received buffer • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• None.

49.2.6.7 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 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 • **Size :** Amount of data to be sent
Return values	• **HAL status**
Notes	• None.

49.2.6.8 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 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 • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• 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.

49.2.6.9 HAL_USART_TransmitReceive_DMA

Function Name	**HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (*USART_HandleTypeDef ** husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function Description	Full-Duplex Transmit Receive an amount of data in non-blocking mode.
Parameters	• **husart :** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData :** Pointer to data transmitted buffer • **pRxData :** Pointer to data received buffer • **Size :** Amount of data to be received
Return values	• **HAL status**
Notes	• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

49.2.6.10 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**
Notes	• None.

49.2.6.11 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**
Notes	• None.

49.2.6.12 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**
Notes	• None.

49.2.6.13 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.
Notes	• None.

49.2.6.14 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.
Notes	• None.

49.2.6.15 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.
Notes	• None.

49.2.6.16 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.
Notes	• None.

49.2.6.17 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.
Notes	• None.

49.2.6.18 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.
Notes	• None.

49.2.6.19 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.
Notes	• None.

49.2.7 Peripheral State and Errors functions

49.2.7.1 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**
Notes	• None.

49.2.7.2 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**
Notes	• None.

49.3 USART Firmware driver defines

49.3.1 USART

USART

USART_Clock

- #define: ***USART_CLOCK_DISABLED ((uint32_t)0x00000000)***

- #define: ***USART_CLOCK_ENABLED ((uint32_t)USART_CR2_CLKEN)***

USART_Clock_Phase

- #define: ***USART_PHASE_1EDGE ((uint32_t)0x00000000)***

- #define: ***USART_PHASE_2EDGE ((uint32_t)USART_CR2_CPHA)***

USART_Clock_Polarity

- #define: ***USART_POLARITY_LOW ((uint32_t)0x00000000)***

- #define: ***USART_POLARITY_HIGH ((uint32_t)USART_CR2_CPOL)***

USART_Flags

- #define: ***USART_FLAG_TXE ((uint32_t)0x00000080)***
- #define: ***USART_FLAG_TC ((uint32_t)0x00000040)***
- #define: ***USART_FLAG_RXNE ((uint32_t)0x00000020)***
- #define: ***USART_FLAG_IDLE ((uint32_t)0x00000010)***
- #define: ***USART_FLAG_ORE ((uint32_t)0x00000008)***
- #define: ***USART_FLAG_NE ((uint32_t)0x00000004)***
- #define: ***USART_FLAG_FE ((uint32_t)0x00000002)***
- #define: ***USART_FLAG_PE ((uint32_t)0x00000001)***

USART_Interrupt_definition

- #define: ***USART_IT_PE ((uint32_t)0x10000100)***
- #define: ***USART_IT_TXE ((uint32_t)0x10000080)***
- #define: ***USART_IT_TC ((uint32_t)0x10000040)***
- #define: ***USART_IT_RXNE ((uint32_t)0x10000020)***

- #define: ***USART_IT_IDLE ((uint32_t)0x10000010)***
- #define: ***USART_IT_LBD ((uint32_t)0x20000040)***
- #define: ***USART_IT_CTS ((uint32_t)0x30000400)***
- #define: ***USART_IT_ERR ((uint32_t)0x30000001)***

USART_Last_Bit

- #define: ***USART_LASTBIT_DISABLE ((uint32_t)0x00000000)***
- #define: ***USART_LASTBIT_ENABLE ((uint32_t)USART_CR2_LBCL)***

USART_Mode

- #define: ***USART_MODE_RX ((uint32_t)USART_CR1_RE)***
- #define: ***USART_MODE_TX ((uint32_t)USART_CR1_TE)***
- #define: ***USART_MODE_TX_RX ((uint32_t)(USART_CR1_TE |USART_CR1_RE))***

USART_NACK_State

- #define: ***USARTNACK_ENABLED ((uint32_t)USART_CR3_NACK)***
- #define: ***USARTNACK_DISABLED ((uint32_t)0x00000000)***

USART_Parity

- #define: ***USART_PARITY_NONE ((uint32_t)0x00000000)***
- #define: ***USART_PARITY_EVEN ((uint32_t)USART_CR1_PCE)***
- #define: ***USART_PARITY_ODD ((uint32_t)(USART_CR1_PCE | USART_CR1_PS))***

USART_Stop_Bits

- #define: ***USART_STOPBITS_1 ((uint32_t)0x00000000)***
- #define: ***USART_STOPBITS_0_5 ((uint32_t)USART_CR2_STOP_0)***
- #define: ***USART_STOPBITS_2 ((uint32_t)USART_CR2_STOP_1)***
- #define: ***USART_STOPBITS_1_5 ((uint32_t)(USART_CR2_STOP_0 | USART_CR2_STOP_1))***

USART_Word_Length

- #define: ***USART_WORDLENGTH_8B ((uint32_t)0x00000000)***
- #define: ***USART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)***

50 HAL WWDG Generic Driver

50.1 WWDG Firmware driver registers structures

50.1.1 WWDG_HandleTypeDef

WWDG_HandleTypeDef is defined in the stm32f4xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_WWDG_StateTypeDef State***

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDef::Instance***
  - Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDef::Init***
  - WWDG required parameters
- ***HAL_LockTypeDef WWDG_HandleTypeDef::Lock***
  - WWDG locking object
- ***__IO HAL_WWDG_StateTypeDef WWDG_HandleTypeDef::State***
  - WWDG communication state

50.1.2 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32f4xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  - Specifies the prescaler. 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 lower than Max_Data = 0x80
- ***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

50.1.3 WWDG_TypeDef

WWDG_TypeDef is defined in the stm32f439xx.h

Data Fields

- ***__IO uint32_t CR***
- ***__IO uint32_t CFR***
- ***__IO uint32_t SR***

Field Documentation

- ***__IO uint32_t WWDG_TypeDef::CR***
  - WWDG Control register, Address offset: 0x00
- ***__IO uint32_t WWDG_TypeDef::CFR***
  - WWDG Configuration register, Address offset: 0x04
- ***__IO uint32_t WWDG_TypeDef::SR***
  - WWDG Status register, Address offset: 0x08

50.2 WWDG Firmware driver API description

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

50.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the WWDG_InitTypeDef and create the associated handle
- DeInitialize the WWDG peripheral
- Initialize the WWDG MSP
- DeInitialize the WWDG MSP
- ***HAL_WWDG_Init()***
- ***HAL_WWDG_DeInit()***
- ***HAL_WWDG_MspInit()***
- ***HAL_WWDG_MspDeInit()***

50.2.2 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- handle WWDG interrupt request.
- ***HAL_WWDG_Start()***
- ***HAL_WWDG_Start_IT()***
- ***HAL_WWDG_Refresh()***

- *HAL_WWDG_IRQHandler()*
- *HAL_WWDG_WakeupCallback()*

50.2.3 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

- *HAL_WWDG_GetState()*

50.2.4 Initialization and de-initialization functions

50.2.4.1 HAL_WWDG_Init

Function Name	**HAL_StatusTypeDef HAL_WWDG_Init (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
Parameters	• **hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL status**
Notes	• None.

50.2.4.2 HAL_WWDG_DeInit

Function Name	**HAL_StatusTypeDef HAL_WWDG_DeInit (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	DeInitializes the WWDG peripheral.
Parameters	• **hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL status**
Notes	• None.

50.2.4.3 HAL_WWDG_MspInit

Function Name	**void HAL_WWDG_MspInit (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	Initializes 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	• None.

50.2.4.4 HAL_WWDG_MspDeInit

Function Name	**void HAL_WWDG_MspDeInit (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	DeInitializes 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	• None.

50.2.5 IO operation functions

50.2.5.1 HAL_WWDG_Start

Function Name	**HAL_StatusTypeDef HAL_WWDG_Start (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	Starts the WWDG.
Parameters	• **hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL status**

Notes	• None.

50.2.5.2 HAL_WWDG_Start_IT

Function Name	**HAL_StatusTypeDef HAL_WWDG_Start_IT (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	Starts the WWDG with interrupt enabled.
Parameters	• **hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL status**
Notes	• None.

50.2.5.3 HAL_WWDG_Refresh

Function Name	**HAL_StatusTypeDef HAL_WWDG_Refresh (*WWDG_HandleTypeDef ** hwwdg, uint32_t Counter)**
Function Description	Refreshes the WWDG.
Parameters	• **hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• **HAL status**
Notes	• None.

50.2.5.4 HAL_WWDG_IRQHandler

Function Name	**void HAL_WWDG_IRQHandler (*WWDG_HandleTypeDef ** hwwdg)**
Function Description	Handles 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 using __HAL_WWDG_ENABLE_IT() macro. 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.

50.2.5.5 HAL_WWDG_WakeupCallback

Function Name		**void HAL_WWDG_WakeupCallback (** ***WWDG_HandleTypeDef **** **hwwdg)**
Function Description		Early Wakeup WWDG callback.
Parameters	•	**hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	•	None.
Notes	•	None.

50.2.6 Peripheral State functions

50.2.6.1 HAL_WWDG_GetState

Function Name		**HAL_WWDG_StateTypeDef HAL_WWDG_GetState (** ***WWDG_HandleTypeDef **** **hwwdg)**
Function Description		Returns the WWDG state.
Parameters	•	**hwwdg :** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	•	**HAL state**

Notes • None.

50.3 WWDG Firmware driver defines

50.3.1 WWDG

WWDG

WWDG_BitAddress_AliasRegion

- #define: ***CFR_BASE (uint32_t)(WWDG_BASE + 0x04)***

WWDG_Flag_definition

- #define: ***WWDG_FLAG_EWIF ((uint32_t)0x0001)***

Early wakeup interrupt flag

WWDG_Interrupt_definition

- #define: ***WWDG_IT_EWI ((uint32_t)WWDG_CFR_EWI)***

WWDG_Prescaler

- #define: ***WWDG_PRESCALER_1 ((uint32_t)0x00000000)***

WWDG counter clock = (PCLK1/4096)/1

- #define: ***WWDG_PRESCALER_2 ((uint32_t)0x00000080)***

WWDG counter clock = (PCLK1/4096)/2

- #define: ***WWDG_PRESCALER_4 ((uint32_t)0x00000100)***

WWDG counter clock = (PCLK1/4096)/4

- #define: ***WWDG_PRESCALER_8 ((uint32_t)0x00000180)***

WWDG counter clock = (PCLK1/4096)/8

51 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 required 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 STM32F4 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 4.4: "Devices supported by 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, Ethernet parameters 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 stm32f4xx_hal_ppp.c/.h and stm32f4xx_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.

Is it possible to use the APIs available in stm32f4xx_ll_ppp.c?

These APIs cannot be used directly because they are internal and offer services to upper layer drivers. As an example stm32f4xx_ll_fmc.c/h driver is used by stm32f4xx_hal_sram.c, stm32f4xx_hal_nor.c, stm32f4xx_hal_nand.c and stm32f4xx_hal_sdram.c drivers.

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() in stm32f4xx_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 stm32f4xx_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.

52 Revision history

Table 16: Document revision history

Date	Revision	Changes
09-May-2014	1	Initial release.

Please Read Carefully