

Description of STM32L0xx HAL drivers

Introduction

STMCube™ 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 STM32CubeL0 for STM32L0 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...
 - 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

1	Acronyms and definitions.....	19
2	Overview of HAL drivers	21
2.1	HAL and user-application files.....	21
2.1.1	HAL driver files	21
2.1.2	User-application files	22
2.2	HAL data structures	24
2.2.1	Peripheral handle structures	24
2.2.2	Initialization and configuration structure	25
2.2.3	Specific process structures	26
2.3	API classification	26
2.4	Devices supported by HAL drivers	27
2.5	HAL drivers rules.....	28
2.5.1	HAL API naming rules	28
2.5.2	HAL general naming rules	29
2.5.3	HAL interrupt handler and callback functions.....	31
2.6	HAL generic APIs.....	31
2.7	HAL extension APIs	33
2.7.1	HAL extension model overview	33
2.7.2	HAL extension model cases	33
2.8	File inclusion model.....	35
2.9	HAL common resources.....	35
2.10	HAL configuration.....	36
2.11	HAL system peripheral handling	37
2.11.1	Clock.....	37
2.11.2	GPIOs.....	38
2.11.3	Cortex NVIC and SysTick timer.....	39
2.11.4	PWR	40
2.11.5	EXTI.....	40
2.11.6	DMA.....	41
2.12	How to use HAL drivers	42
2.12.1	HAL usage models	42
2.12.2	HAL initialization	43
2.12.3	HAL IO operation process	46
2.12.4	Timeout and error management.....	49
3	HAL Generic Driver	53

3.1	HAL Firmware driver introduction	53
3.2	HAL Firmware driver API description	53
3.2.1	How to use this driver	53
3.2.2	Initialization and de-initialization functions	53
3.2.3	HAL Control functions.....	53
3.3	HAL Firmware driver defines.....	63
3.3.1	HAL.....	63
4	HAL ADC Generic Driver.....	68
4.1	ADC Firmware driver introduction	68
4.2	ADC Firmware driver registers structures	68
4.2.1	ADC_HandleTypeDef.....	68
4.2.2	ADC_InitTypeDef.....	68
4.2.3	ADC_ChannelConfTypeDef	70
4.2.4	ADC_AnalogWDGConfTypeDef.....	70
4.2.5	ADC_Common_TypeDef.....	71
4.2.6	ADC_TypeDef	71
4.3	ADC Firmware driver API description.....	73
4.3.1	ADC specific features	73
4.3.2	How to use this driver	73
4.3.3	Initialization and de-initialization functions	74
4.3.4	IO operation functions	74
4.3.5	Peripheral Control functions	74
4.3.6	ADC Peripheral State functions.....	75
4.4	ADC Firmware driver defines	83
4.4.1	ADC	83
5	HAL ADC Extension Driver	95
5.1	ADCEx Firmware driver introduction	95
5.2	ADCEx Firmware driver API description	95
5.2.1	ADC specific features	95
5.2.2	How to use this driver	95
5.2.3	ADC Extended features functions	95
5.3	ADCEx Firmware driver defines	97
5.3.1	ADCEx	97
6	HAL COMP Generic Driver.....	98
6.1	COMP Firmware driver introduction	98
6.2	COMP Firmware driver registers structures	98

Contents	UM1749
6.2.1 COMP_HandleTypeDef.....	98
6.2.2 COMP_InitTypeDef	98
6.2.3 COMP_TypeDef	99
6.3 COMP Firmware driver API description	99
6.4 COMP Firmware driver defines	104
6.4.1 COMP.....	104
7 HAL CORTEX Generic Driver.....	108
7.1 CORTEX Firmware driver introduction	108
7.2 CORTEX Firmware driver API description	108
7.2.1 How to use this driver.....	108
7.2.2 Initialization and de-initialization functions	109
7.2.3 Peripheral Control functions	109
7.3 CORTEX Firmware driver defines	113
7.3.1 CORTEX.....	113
8 HAL CRC Generic Driver.....	114
8.1 CRC Firmware driver introduction	114
8.2 CRC Firmware driver registers structures	114
8.2.1 CRC_InitTypeDef	114
8.2.2 CRC_HandleTypeDef.....	114
8.2.3 CRC_TypeDef	115
8.3 CRC Firmware driver API description	116
8.3.1 Initialization and de-initialization functions	116
8.3.2 Peripheral Control functions	116
8.3.3 Peripheral State functions	116
8.4 CRC Firmware driver defines	120
8.4.1 CRC.....	120
9 HAL CRC Extension Driver	123
9.1 CRCEx Firmware driver introduction	123
9.2 CRCEx Firmware driver API description	123
9.2.1 CRC Extended features functions	123
9.3 CRCEx Firmware driver defines.....	124
9.3.1 CRCEx.....	124
10 HAL CRYP Generic Driver.....	125
10.1 CRYP Firmware driver introduction	125
10.2 CRYP Firmware driver registers structures	125
10.2.1 CRYP_HandleTypeDef.....	125

10.2.2	CRYP_InitTypeDef	126
10.2.3	AES_TypeDef.....	126
10.3	CRYP Firmware driver API description	127
10.3.1	Initialization and de-initialization functions	127
10.3.2	AES processing functions	127
10.3.3	DMA callback functions	128
10.3.4	CRYP IRQ handler management.....	128
10.3.5	Peripheral State functions	128
10.4	CRYP Firmware driver defines.....	140
10.4.1	CRYP	140
11	HAL DAC Generic Driver.....	142
11.1	DAC Firmware driver registers structures	142
11.1.1	DAC_HandleTypeDef.....	142
11.1.2	DAC_ChannelConfTypeDef	142
11.1.3	DAC_TypeDef	143
11.2	DAC Firmware driver API description.....	143
11.2.1	Initialization and de-initialization functions	143
11.2.2	IO operation functions	144
11.2.3	Peripheral Control functions	144
11.2.4	DAC Peripheral State functions.....	144
11.3	DAC Firmware driver defines	150
11.3.1	DAC	150
12	HAL DAC Extension Driver	152
12.1	DACEx Firmware driver introduction	152
12.2	DACEx Firmware driver API description	152
12.2.1	Peripheral Control functions	152
12.3	DACEx Firmware driver defines	154
12.3.1	DACEx	154
13	HAL DMA Generic Driver	157
13.1	DMA Firmware driver registers structures	157
13.1.1	DMA_HandleTypeDef.....	157
13.1.2	DMA_InitTypeDef	157
13.1.3	DMA_Channel_TypeDef.....	158
13.1.4	DMA_TypeDef	159
13.1.5	DMA_Request_TypeDef.....	159
13.2	DMA Firmware driver API description	159

13.2.1	Initialization and de-initialization functions	160
13.2.2	IO operation functions	160
13.2.3	Peripheral State functions	160
13.3	DMA Firmware driver defines.....	164
13.3.1	DMA.....	164
14	HAL FLASH Generic Driver.....	171
14.1	FLASH Firmware driver introduction	171
14.2	FLASH Firmware driver registers structures	171
14.2.1	FLASH_EraseInitTypeDef	171
14.2.2	FLASH_OBProgramInitTypeDef	171
14.2.3	FLASH_ProcessTypeDef	172
14.2.4	FLASH_TypeDef	172
14.3	FLASH Firmware driver API description.....	173
14.3.1	FLASH peripheral features	173
14.3.2	How to use this driver	173
14.3.3	Programming operation functions	174
14.3.4	Option Bytes Programming functions.....	174
14.3.5	Peripheral Control functions	175
14.3.6	Peripheral Errors functions	175
14.4	FLASH Firmware driver defines	179
14.4.1	FLASH	179
15	HAL FLASH Extension Driver.....	186
15.1	FLASHEx Firmware driver introduction	186
15.2	FLASHEx Firmware driver registers structures	186
15.2.1	FLASH_AdvOBProgramInitTypeDef	186
15.3	FLASHEx Firmware driver API description.....	186
15.3.1	Flash peripheral Extended features	186
15.3.2	How to use this driver	187
15.3.3	Extended Features functions.....	187
15.3.4	DATA EEPROM Programming functions	187
15.4	FLASHEx Firmware driver defines	191
15.4.1	FLASHEx	191
16	HAL FLASH Ram Function Driver	196
16.1	FLASHRamfunc Firmware driver introduction	196
16.2	FLASHRamfunc Firmware driver API description	196
16.2.1	ramfunc functions	196

16.3	FLASHRamfunc Firmware driver defines	197
16.3.1	FLASHRamfunc.....	197
17	HAL GPIO Generic Driver.....	198
17.1	GPIO Firmware driver introduction	198
17.2	GPIO Firmware driver registers structures	198
17.2.1	GPIO_InitTypeDef	198
17.2.2	GPIO_TypeDef	198
17.3	GPIO Firmware driver API description	199
17.3.1	GPIO Peripheral features	199
17.3.2	How to use this driver	200
17.3.3	Initialization and de-initialization functions	201
17.3.4	IO operation functions	201
17.4	GPIO Firmware driver defines.....	204
17.4.1	GPIO.....	204
18	HAL GPIO Extension Driver.....	208
18.1	GPIOEx Firmware driver defines.....	208
18.1.1	GPIOEx	208
19	HAL I2C Generic Driver.....	213
19.1	I2C Firmware driver introduction	213
19.2	I2C Firmware driver registers structures	213
19.2.1	I2C_HandleTypeDef	213
19.2.2	I2C_InitTypeDef.....	214
19.2.3	I2C_TypeDef	214
19.3	I2C Firmware driver API description.....	215
19.3.1	How to use this driver	215
19.3.2	Initialization and de-initialization functions	218
19.3.3	IO operation functions	219
19.3.4	Peripheral State and Errors functions	220
19.4	I2C Firmware driver defines	234
19.4.1	I2C	234
20	HAL I2C Extension Driver	239
20.1	I2CEx Firmware driver introduction	239
20.2	I2CEx Firmware driver API description	239
20.2.1	Peripheral Control functions	239
20.3	I2CEx Firmware driver defines	241
20.3.1	I2CEx	241

21 HAL I2S Generic Driver	242
21.1 I2S Firmware driver introduction	242
21.2 I2S Firmware driver registers structures	242
21.2.1 I2S_HandleTypeDef	242
21.2.2 I2S_InitTypeDef	242
21.2.3 SPI_TypeDef	243
21.3 I2S Firmware driver API description.....	244
21.3.1 How to use this driver	244
21.3.2 Initialization and de-initialization functions	246
21.3.3 IO operation functions	247
21.3.4 Peripheral State and Errors functions	247
21.4 I2S Firmware driver defines	256
21.4.1 I2S	256
22 HAL IRDA Generic Driver.....	260
22.1 IRDA Firmware driver introduction	260
22.2 IRDA Firmware driver registers structures	260
22.2.1 IRDA_HandleTypeDef	260
22.2.2 IRDA_InitTypeDef	261
22.2.3 USART_TypeDef	261
22.3 IRDA Firmware driver API description.....	262
22.3.1 Initialization and Configuration functions	262
22.3.2 IO operation functions	263
22.3.3 Peripheral Control functions	264
22.4 IRDA Firmware driver defines	269
22.4.1 IRDA	269
23 HAL IRDA Extension Driver	275
23.1 IRDAEx Firmware driver defines	275
23.1.1 IRDAEx	275
24 HAL IWDG Generic Driver.....	276
24.1 IWDG Firmware driver introduction	276
24.2 IWDG Firmware driver registers structures	276
24.2.1 IWDG_HandleTypeDef	276
24.2.2 IWDG_InitTypeDef	276
24.2.3 IWDG_TypeDef	277
24.3 IWDG Firmware driver API description	277
24.3.1 IWDG Generic features	277

24.3.2	How to use this driver	278
24.3.3	Initialization and de-initialization functions	278
24.3.4	IO operation functions	279
24.3.5	Peripheral State functions	279
24.4	IWDG Firmware driver defines	281
24.4.1	IWDG	281
25	HAL LCD Generic Driver	283
25.1	LCD Firmware driver introduction.....	283
25.2	LCD Firmware driver registers structures.....	283
25.2.1	LCD_HandleTypeDef	283
25.2.2	LCD_InitTypeDef	283
25.2.3	LCD_TypeDef	284
25.3	LCD Firmware driver API description	285
25.3.1	How to use this driver	285
25.3.2	Initialization and Configuration functions.....	286
25.3.3	IO operation functions	286
25.3.4	Peripheral State functions	286
25.4	LCD Firmware driver defines.....	290
25.4.1	LCD.....	290
26	HAL LPTIM Generic Driver.....	300
26.1.1	LPTIM Firmware driver introduction	300
26.2	LPTIM Firmware driver registers structures	300
26.2.1	LPTIM_ClockConfigTypeDef	300
26.2.2	LPTIM_ULPClockConfigTypeDef	300
26.2.3	LPTIM_TriggerConfigTypeDef	301
26.2.4	LPTIM_InitTypeDef.....	301
26.2.5	LPTIM_HandleTypeDef	302
26.2.6	LPTIM_TypeDef	302
26.3	LPTIM Firmware driver API description.....	303
26.3.1	Initialization and de-initialization functions	303
26.3.2	LPTIM Start Stop operation functions	303
26.3.3	LPTIM Read operation functions	304
26.3.4	LPTIM IRQ handler.....	304
26.3.5	Peripheral State functions	305
26.4	LPTIM Firmware driver defines	319
26.4.1	LPTIM	319

27 HAL PCD Generic Driver	325
27.1 PCD Firmware driver introduction	325
27.2 PCD Firmware driver registers structures	325
27.2.1 PCD_EPTTypeDef.....	325
27.2.2 PCD_InitTypeDef.....	326
27.2.3 PCD_HandleTypeDef	327
27.2.4 USB_TypeDef.....	327
27.3 PCD Firmware driver API description.....	329
27.3.1 How to use this driver	329
27.3.2 Initialization and de-initialization functions	330
27.3.3 IO operation functions	330
27.3.4 Peripheral Control functions	330
27.3.5 Peripheral State functions	331
27.4 PCD Firmware driver defines	342
27.4.1 PCD	342
28 HAL PCD Extension Driver	344
28.1.1 PCDEx Firmware driver introduction	344
28.2 PCDEx Firmware driver API description	344
28.2.1 Peripheral extended features functions	344
28.3 PCDEx Firmware driver defines	344
28.3.1 PCDEx	344
29 HAL PWR Generic Driver	345
29.1.1 PWR Firmware driver introduction	345
29.2 PWR Firmware driver registers structures	345
29.2.1 PWR_PVDTTypeDef	345
29.2.2 PWR_TypeDef.....	345
29.3 PWR Firmware driver API description.....	345
29.3.1 Initialization and de-initialization functions	346
29.3.2 Peripheral Control functions	346
29.4 PWR Firmware driver defines	355
29.4.1 PWR	355
30 HAL PWR Extension Driver	358
30.1.1 PWREx Firmware driver introduction	358
30.2 PWREx Firmware driver API description.....	358
30.2.1 Peripheral extended features functions	358
30.3 PWREx Firmware driver defines	360

30.3.1	PWREx	360
31	HAL RCC Generic Driver.....	361
31.1.1	RCC Firmware driver introduction	361
31.2	RCC Firmware driver registers structures	361
31.2.1	RCC_PLLInitTypeDef	361
31.2.2	RCC_ClkInitTypeDef	361
31.2.3	RCC_OscInitTypeDef	362
31.2.4	RCC_TypeDef	363
31.3	RCC Firmware driver API description	364
31.3.1	RCC specific features	364
31.3.2	Initialization and de-initialization functions	365
31.3.3	Peripheral Control functions	366
31.4	RCC Firmware driver defines	372
31.4.1	RCC	372
32	HAL RCC Extension Driver	395
32.1.1	RCCEEx Firmware driver introduction	395
32.2	RCCEEx Firmware driver registers structures	395
32.2.1	RCC_CRSInitTypeDef	395
32.2.2	RCC_CRSSynchroInfoTypeDef	396
32.2.3	RCC_PерiphCLKInitTypeDef	396
32.2.4	CRS_TypeDef	397
32.3	RCCEEx Firmware driver API description	397
32.3.1	Extended Peripheral Control functions	398
32.4	RCCEEx Firmware driver defines	401
32.4.1	RCCEEx	401
33	HAL RNG Generic Driver.....	423
33.1.1	RNG Firmware driver introduction	423
33.2	RNG Firmware driver registers structures	423
33.2.1	RNG_HandleTypeDef	423
33.2.2	RNG_TypeDef	423
33.3	RNG Firmware driver API description	424
33.3.1	How to use this driver	424
33.3.2	Initialization and de-initialization functions	424
33.3.3	Peripheral Control functions	424
33.3.4	Peripheral State functions	424
33.4	RNG Firmware driver defines	428

33.4.1	RNG.....	428
34	HAL RTC Generic Driver	430
34.1.1	RTC Firmware driver introduction	430
34.2	RTC Firmware driver registers structures	430
34.2.1	RTC_HandleTypeDef	430
34.2.2	RTC_InitTypeDef.....	430
34.2.3	RTC_DateTypeDef	431
34.2.4	RTC_TimeTypeDef.....	432
34.2.5	RTC_AlarmTypeDef	432
34.2.6	RTC_TypeDef.....	433
34.3	RTC Firmware driver API description.....	435
34.3.1	Backup Domain Operating Condition.....	435
34.3.2	Backup Domain Reset.....	435
34.3.3	Backup Domain Access.....	435
34.3.4	How to use this driver	435
34.3.5	RTC and low power modes	437
34.3.6	Initialization and de-initialization functions	437
34.3.7	RTC Time and Date functions	438
34.3.8	RTC Alarm functions	438
34.3.9	Peripheral Control functions	438
34.3.10	Peripheral State functions	438
34.4	RTC Firmware driver defines	447
34.4.1	RTC	447
35	HAL RTC Extension Driver	455
35.1.1	RTCEEx Firmware driver introduction	455
35.2	RTCEEx Firmware driver registers structures	455
35.2.1	RTC_TamperTypeDef	455
35.3	RTCEEx Firmware driver API description.....	456
35.3.1	RTCTimeStamp and Tamper functions.....	456
35.4	RTCEEx Firmware driver defines	470
35.4.1	RTCEEx	470
36	HAL SMARTCARD Generic Driver.....	477
36.1	SMARTCARD Firmware driver introduction	477
36.2	SMARTCARD Firmware driver registers structures	477
36.2.1	SMARTCARD_HandleTypeDef.....	477
36.2.2	SMARTCARD_InitTypeDef	478
36.2.3	SMARTCARD_AdvFeatureInitTypeDef.....	479

36.2.4	USART_TypeDef.....	480
36.3	SMARTCARD Firmware driver API description.....	481
36.3.1	Initialization and Configuration functions.....	481
36.3.2	IO operation functions	482
36.3.3	Peripheral State functions	482
36.4	SMARTCARD Firmware driver defines	488
36.4.1	SMARTCARD.....	488
37	HAL SMARTCARD Extension Driver	497
37.1	SMARTCARDEX Firmware driver introduction	497
37.2	SMARTCARDEX Firmware driver API description	497
37.2.1	How to use this driver	497
37.2.2	Peripheral Control functions	497
37.3	SMARTCARDEX Firmware driver defines	499
37.3.1	SMARTCARDEX.....	499
38	HAL SMBUS Generic Driver.....	500
38.1	SMBUS Firmware driver introduction	500
38.2	SMBUS Firmware driver registers structures	500
38.2.1	SMBUS_InitTypeDef	500
38.2.2	SMBUS_HandleTypeDef.....	501
38.2.3	I2C_HandleTypeDef	502
38.3	SMBUS Firmware driver API description	503
38.3.1	Initialization and de-initialization functions	503
38.3.2	IO operation functions	503
38.3.3	Peripheral State and Errors functions	504
38.4	SMBUS Firmware driver defines	514
38.4.1	SMBUS	514
39	HAL SPI Generic Driver.....	521
39.1	SPI Firmware driver introduction	521
39.2	SPI Firmware driver registers structures	521
39.2.1	SPI_HandleTypeDef.....	521
39.2.2	SPI_InitTypeDef	522
39.2.3	SPI_TypeDef	523
39.3	SPI Firmware driver API description	524
39.3.1	How to use this driver	524
39.3.2	Initialization and de-initialization functions	525
39.3.3	IO operation functions	525

Contents	UM1749
39.3.4 Peripheral State and Errors functions	526
39.4 SPI Firmware driver defines	534
39.4.1 SPI	534
40 HAL TIM Generic Driver	538
40.1 TIM Firmware driver introduction.....	538
40.2 TIM Firmware driver registers structures.....	538
40.2.1 TIM_HandleTypeDef	538
40.2.2 TIM_Base_InitTypeDef.....	538
40.2.3 TIM_OC_InitTypeDef.....	539
40.2.4 TIM_IC_InitTypeDef	539
40.2.5 TIM_OnePulse_InitTypeDef	540
40.2.6 TIM_ClockConfigTypeDef	541
40.2.7 TIM_ClearInputConfigTypeDef.....	541
40.2.8 TIM_SlaveConfigTypeDef	542
40.2.9 TIM_Encoder_InitTypeDef	542
40.2.10 TIM_TypeDef.....	543
40.3 TIM Firmware driver API description	546
40.3.1 TIMER Generic features.....	546
40.3.2 How to use this driver	546
40.3.3 Initialization and de-initialization functions	547
40.3.4 IO operation functions	547
40.3.5 Peripheral Control functions	549
40.3.6 TIM Callbacks functions	549
40.3.7 Peripheral State functions	549
40.3.8 IRQ handler management.....	550
40.4 TIM Firmware driver defines.....	587
40.4.1 TIM.....	587
41 HAL TIM Extension Driver.....	604
41.1 TIMEEx Firmware driver introduction	604
41.2 TIMEEx Firmware driver registers structures.....	604
41.2.1 TIM_MasterConfigTypeDef	604
41.3 TIMEEx Firmware driver API description	604
41.3.1 TIM specific features integration	604
41.3.2 How to use this driver.....	604
41.3.3 Peripheral Control functions	605
41.4 TIMEEx Firmware driver defines	607
41.4.1 TIMEEx	607

42 HAL TSC Generic Driver	611
42.1 TSC Firmware driver introduction.....	611
42.2 TSC Firmware driver registers structures.....	611
42.2.1 TSC_HandleTypeDef	611
42.2.2 TSC_InitTypeDef	611
42.2.3 TSC_IOConfigTypeDef.....	612
42.2.4 TSC_TypeDef.....	613
42.3 TSC Firmware driver API description	614
42.3.1 TSC specific features	614
42.3.2 How to use this driver.....	614
42.3.3 Initialization and de-initialization functions	615
42.4 TSC Firmware driver defines.....	616
42.4.1 TSC.....	616
43 HAL UART Generic Driver.....	628
43.1 UART Firmware driver introduction	628
43.2 UART Firmware driver registers structures	628
43.2.1 UART_HandleTypeDef.....	628
43.2.2 UART_AdvFeatureInitTypeDef.....	629
43.2.3 UART_InitTypeDef	630
43.2.4 USART_TypeDef.....	630
43.3 UART Firmware driver API description	632
43.3.1 Initialization and Configuration functions.....	632
43.3.2 IO operation functions	632
43.3.3 Peripheral Control functions	633
43.4 UART Firmware driver defines	644
43.4.1 UART	644
44 HAL UART Extension Driver.....	657
44.1 UARTEEx Firmware driver introduction	657
44.2 UARTEEx Firmware driver registers structures	657
44.2.1 UART_WakeUpTypeDef	657
44.3 UARTEEx Firmware driver API description	657
44.3.1 Initialization and Configuration functions.....	657
44.3.2 Peripheral Control funtions	657
44.4 UARTEEx Firmware driver defines	661
44.4.1 UARTEEx.....	661
45 HAL USART Generic Driver	663

Contents	UM1749
45.1 USART Firmware driver introduction.....	663
45.2 USART Firmware driver registers structures.....	663
45.2.1 USART_HandleTypeDef	663
45.2.2 USART_InitTypeDef	664
45.2.3 USART_TypeDef.....	665
45.3 USART Firmware driver API description	666
45.3.1 Initialization and de-initialization functions	666
45.3.2 IO operation functions	666
45.3.3 Peripheral State functions	667
45.4 USART Firmware driver defines.....	677
45.4.1 USART.....	677
46 HAL USART Extension Driver	682
46.1 USARTEx Firmware driver defines	682
46.1.1 USARTEx	682
47 HAL WWDG Generic Driver	683
47.1 WWDG Firmware driver introduction.....	683
47.2 WWDG Firmware driver registers structures.....	683
47.2.1 WWDG_HandleTypeDef	683
47.2.2 WWDG_InitTypeDef	683
47.2.3 WWDG_TypeDef.....	684
47.3 WWDG Firmware driver API description	684
47.3.1 WWDG specific features	684
47.3.2 How to use this driver	685
47.3.3 Initialization and de-initialization functions	685
47.3.4 IO operation functions	685
47.3.5 Peripheral State functions	686
47.4 WWDG Firmware driver defines.....	690
47.4.1 WWDG.....	690

List of tables

Table 1: Acronyms and definitions	19
Table 2: HAL drivers files	21
Table 3: User-application files	22
Table 4: APis classification	27
Table 5: List of devices supported by HAL drivers	27
Table 6: HAL API naming rules	28
Table 7: Macros handling interrupts and specific clock configurations	30
Table 8: Callback functions	31
Table 9: HAL generic APIs	32
Table 10: HAL extension APIs	33
Table 11: Define statements used for HAL configuration	36
Table 12: Description of GPIO_InitTypeDef structure	38
Table 13: Description of EXTI configuration macros	40
Table 14: MSP functions	45
Table 15: Timeout values	49

List of figures

Figure 1: Example of project template	24
Figure 2: Adding device-specific functions	33
Figure 3: Adding family-specific functions	34
Figure 4: Adding new peripherals	34
Figure 5: Updating existing APIs	34
Figure 6: File inclusion model	35
Figure 7: HAL driver model	43

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
COMP	Comparator
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
LPTIM	Low Power Timer
MSP	MCU Specific Package
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer

Acronym	Definition
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch Sensing Controller
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 a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: USART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
 - Fully re-entrant 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

HAL drivers are composed of the following set of files:

Table 2: HAL drivers files

File	Description
<i>stm32l0xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32l0xx_hal_adc.c, stm32l0xx_hal_irda.c, ...</i>
<i>stm32l0xx_hal_ppp.h</i>	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. <i>Example: stm32l0xx_hal_adc.h, stm32l0xx_hal_irda.h, ...</i>

File	Description
<i>stm32l0xx_hal_ppp_ex.c</i>	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. <i>Example: stm32l0xx_hal_adc_ex.c, stm32l0xx_hal_dma_ex.c, ...</i>
<i>stm32l0xx_hal_ppp_ex.h</i>	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs <i>Example: stm32l0xx_hal_adc_ex.h, stm32l0xx_hal_dma_ex.h, ...</i>
<i>stm32l0xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32l0xx_hal.h</i>	<i>stm32l0xx_hal.c</i> header file
<i>stm32l0xx_hal_msp_template.c</i>	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l0xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32l0xx_hal_def.h</i>	Common HAL resources such as common define statements, enumerations, structures and macros.

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
<i>system_stm32l0xx.c</i>	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.
<i>startup_stm32l0xx.s</i>	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
<i>stm32l0xx_flash.icf (optional)</i>	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32l0xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l0xx_hal_conf.h</i>	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
<i>stm32l0xx_it.c/h</i>	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in <i>stm32l0xx_hal.c</i>) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
<i>main.c/h</i>	This file contains the main program routine, in particular, call to HAL_Init(), assert_failed() implementation, system clock configuration, HAL peripheral 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.

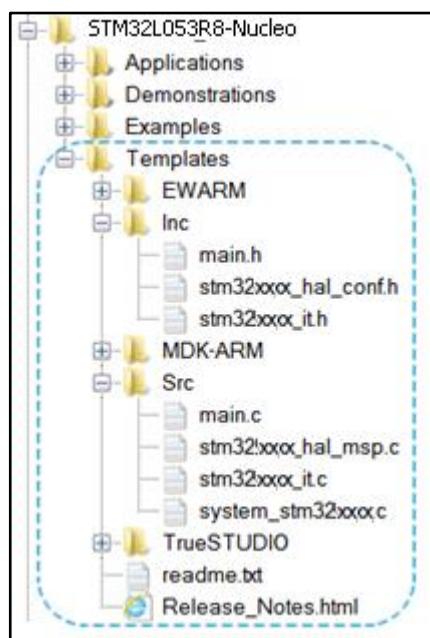
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- 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 simultaneously.

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

The peripheral handle is used for the following purposes:

- Multi instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
    USART_TypeDef *Instance; /* USART registers base address */
    USART_InitTypeDef Init; /* Usart communication parameters */
    uint8_t *pTxBuffPtr; /* Pointer to Usart Tx transfer Buffer */
    uint16_t TxXferSize; /* Usart Tx Transfer size */
    __IO uint16_t TxXferCount; /* Usart Tx Transfer Counter */
    uint8_t *pRxBuffPtr; /* Pointer to Usart Rx transfer Buffer */
}
```

```

    uint16_t RxXferSize; /* Usart Rx Transfer size */
    _IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
    DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
    DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
    HAL_LockTypeDef Lock; /* Locking object */
    IO_HAL_USART_StateTypeDef State; /* Usart communication state */
    _IO HAL_USART_ErrorTypeDef ErrorCode; /* USART Error code */
}USART_HandleTypeDef;

```



1) The multi-instance feature implies that all the APIs used in the application are re-entrant and avoid using global variables because subroutines 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 whether the Receive or Transmit mode is enabled or
                    disabled.*/
    uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled
                        or disabled.*/
    uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or
                           disabled.*/
}

```

```
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_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
```

- **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(STM32L051xx) && !defined(STM32L061xx)
void HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit);
void HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void);
RCC_CRSStatusTypeDef HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout);
#endif /* !(STM32L051xx) && !(STM32L061xx) */
```

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	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx
stm32l0xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_cryp.c	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_dac.c	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_dac_ex.c	No	Yes	Yes	No	Yes	No
stm32l0xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash_ex.c	No	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash_ramfunc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2s.c	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx
stm32l0xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_lcd.c	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_pcd.c	No	Yes	Yes	No	No	No
stm32l0xx_hal_pcd_ex.c	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rng.c	No	No	Yes	No	No	Yes
stm32l0xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_tsc.c	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_wwdg.c	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	stm32l0xx_hal_ppp (c/h)	stm32l0xx_hal_ppp_ex (c/h)	stm32l0xx_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

	Generic	Family specific	Device specific
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32L0xx reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in stm32l0xxx.h header file. stm32l0xxx.h corresponds to stm32l051xx.h, stm32l052xx.h, stm32l053xx.h, stm32l061xx.h, stm32l062xx.h or stm32l063xx.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_DelInit, e.g. TIM_DelInit.
- 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_Start() 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
<code>_HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Enables a specific peripheral interrupt
<code>_HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Disables a specific peripheral interrupt
<code>_HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)</code>	Gets a specific peripheral interrupt status
<code>_HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)</code>	Clears a specific peripheral interrupt status
<code>_HAL_PPP_GET_FLAG (__HANDLE__, __ FLAG__)</code>	Gets a specific peripheral flag status
<code>_HAL_PPP_CLEAR_FLAG (__HANDLE__, __ FLAG__)</code>	Clears a specific peripheral flag status
<code>_HAL_PPP_ENABLE(__HANDLE__)</code>	Enables a peripheral
<code>_HAL_PPP_DISABLE(__HANDLE__)</code>	Disables a peripheral
<code>_HAL_PPP_XXXX (__HANDLE__, __PARAM__)</code>	Specific PPP HAL driver macro
<code>_HAL_PPP_GET_IT_SOURCE (__HANDLE__, __ INTERRUPT__)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the `stm320xx_hal_cortex.c` file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example : `STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)"`.
- The PPP handles are valid before using the `HAL_PPP_Init()` API. The init function performs a check before modifying the handle fields.

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

- The macros defined below are used:
 - Conditional macro: `#define ABS(x) (((x) > 0) ? (x) : -(x))`
 - Pseudo-code macro (multiple instructions macro):

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

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 stm32l0xx_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_MspDelInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8: Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DelInit()	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_DelInit()
- **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_DelInit()*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
<i>Initialization group</i>	<i>HAL_ADC_Init()</i>	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	<i>HAL_ADC_DeInit()</i>	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
<i>IO operation group</i>	<i>HAL_ADC_Start ()</i>	This function starts ADC conversions when the polling method is used
	<i>HAL_ADC_Stop ()</i>	This function stops ADC conversions when the polling method is used
	<i>HAL_ADC_PollForConversion()</i>	This function allows waiting for the end of conversions when the polling method is used. In this case, a timeout value is specified by the user according to the application.
	<i>HAL_ADC_Start_IT()</i>	This function starts ADC conversions when the interrupt method is used
	<i>HAL_ADC_Stop_IT()</i>	This function stops ADC conversions when the interrupt method is used
	<i>HAL_ADC_IRQHandler()</i>	This function handles ADC interrupt requests
	<i>HAL_ADC_ConvCpltCallback()</i>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	<i>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
<i>Control group</i>	<i>HAL_ADC_ConfigChannel()</i>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	<i>HAL_ADC_AnalogWDGConfig</i>	This function configures the analog watchdog for the selected ADC
<i>State and Errors group</i>	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function 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, `stm32l0xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32l0xx_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
<code>HAL_ADCEx_CalibrationStart()</code>	This function is used to start the automatic ADC calibration
<code>HAL_ADCEx_Calibration_GetValue()</code>	This function is used to get the ADC calibration factor
<code>HAL_ADCEx_Calibration_SetValue()</code>	This function is used to set the calibration factor to overwrite automatic conversion result

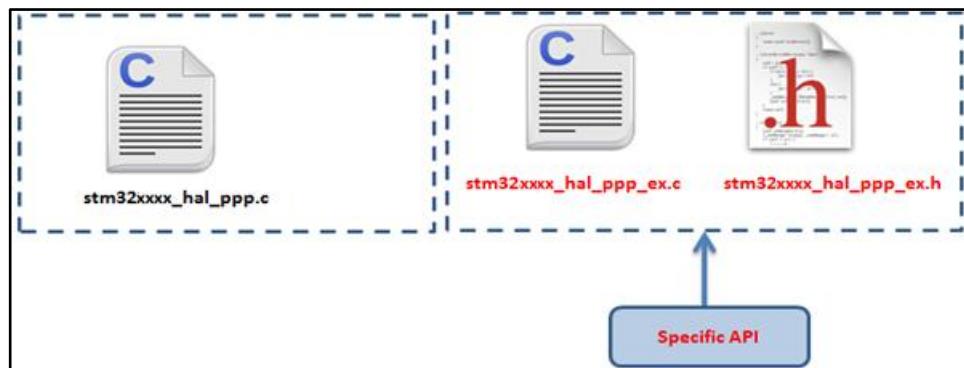
2.7.2 HAL extension model cases

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 `stm32l0xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEX_Function()`.

Figure 2: Adding device-specific functions



Example: `stm32l0xx_hal_rcc_ex.c/h`

```
#if !defined(STM32L051xx) && !defined(STM32L061xx)
void HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit);
void HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void);
void HAL_RCCEx_CRSGetSynchronizationInfo(RCC_CRSSynchroInfoTypeDef *pSynchroInfo);
RCC_CRSStatusTypeDef HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout);
#endif /* !(STM32L051xx) && !(STM32L061xx) */
```

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

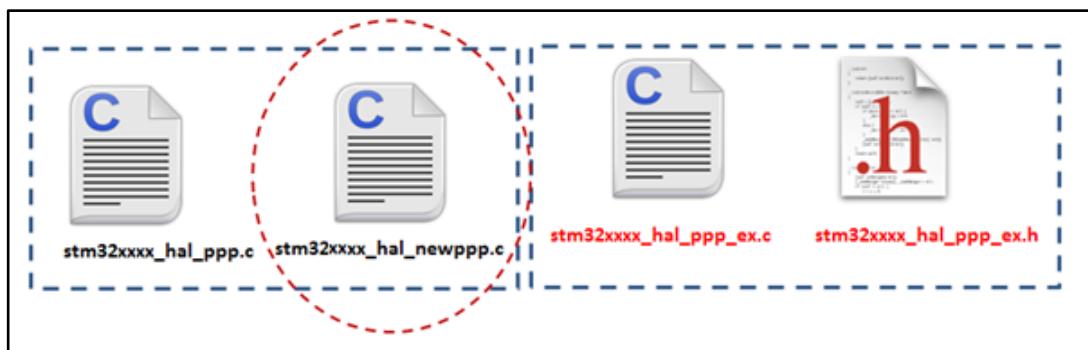


Case3: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32l0xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lx_xx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

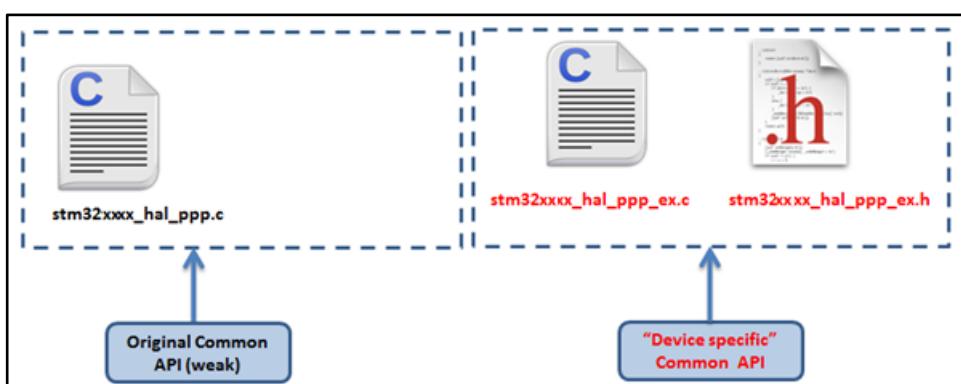


Example: `stm32l0xx_hal_lcd.c/h`

Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32l0xx_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:

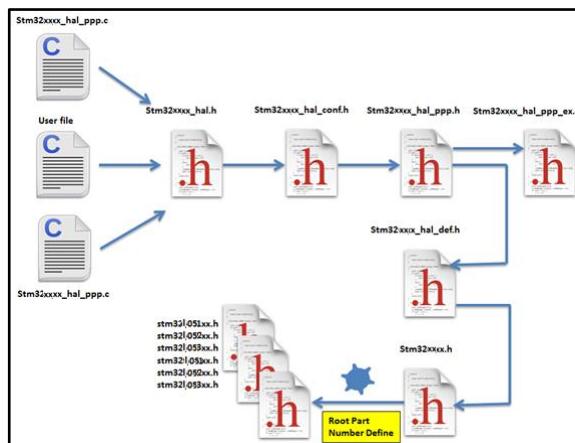
```
#if defined (STM32L051xx)
typedef struct
{
(...)

}PPP_InitTypeDef;
#endif /* STM32L051xx */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32l0xx_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 stm32l0xx_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 *stm32l0xx_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 `stm32l0xx_hal_def.h` file calls the `stm32l0xx.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, `stm32l0xx_hal_conf.h`, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 11: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
MSI_VALUE	Defines the Internal Multiple Speed oscillator (MSI) value expressed in Hz.	2 000 000 (Hz)
VDD_VALUE	VDD value	3300 (mV)

Configuration item	Description	Default Value
USERTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE
BUFFER_CACHE_ENABLE	Enables buffer cache	FALSE



The `stm32l0xx_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 `stm32l0xx_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 of Flash memory wait states
 - Updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, USB...). In this case, the clock configuration is performed by an extended API defined in `stm32l0xx_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 retrieving various clock configurations (system clock, HCLK, PCLK1, PCLK2, ...)
- MCO and CSS configuration functions

A set of macros are defined in `stm32l0xx_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 stm32l0xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 12: Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: <ul style="list-style-type: none"> • <u>GPIO mode</u> <ul style="list-style-type: none"> – 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 • <u>External Interrupt Mode</u> <ul style="list-style-type: none"> – 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 • <u>External Event Mode</u> <ul style="list-style-type: none"> – 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

Structure field	Description
Alternate	<p>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.</p>  <p>Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.</p>

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

- Configuring USART1 Tx (PA9, mapped on AF4) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
GPIO_InitStruct.Alternate = GPIO_AF4_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32l0xx_hal_cortex.c, provides APIs to handle NVIC and Systick. The supported APIs include:

- HAL_NVIC_SetPriority()
- HAL_NVIC_EnableIRQ() / HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ() / HAL_NVIC_ClearPendingIRQ()
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()

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 `stm32l0xx_hal_pwr_ex`. Here are a few examples (the list is not exhaustive)

- Ultra low power mode control
 - HAL_PWREx_EnableUltraLowPower() / HAL_PWREx_DisableUltraLowPower()
 - HAL_PWREx_EnableLowPowerRunMode() / HAL_PWREx_DisableLowPowerRunMode()

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
<code>PPP_EXTI_LINE_FUNCTION</code>	Defines the EXTI line connected to the internal peripheral. Example: <code>#define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */</code>
<code>_HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)</code>	Enables a given EXTI line Example: <code>_HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)</code>	Disables a given EXTI line. Example: <code>_HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)</code>

Macros	Description
<code>_HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)</code>	Gets a given EXTI line interrupt flag pending bit status. Example: <code>_HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)</code>	Clears a given EXTI line interrupt flag pending bit. Example; <code>_HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_GENERATE_SWIT(__EXTI_LINE__)</code>	Generates a software interrupt for a given EXTI line. Example: <code>_HAL_PVD_EXTI_GENERATE_SWIT(PWR_EXTI_LINE_PVD)</code>

If the EXTI interrupt mode is selected, the user application must call `HAL_PPP_FUNCTION_IRQHandler()` (for example `HAL_PWR_PVD_IRQHandler()`), from `stm32lx_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 Channels (except for internal SRAM/FLASH memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given 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
- Channels 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 configured. 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: enables the specified DMA Channels.
- __HAL_DMA_DISABLE: disables the specified DMA Channels.
- __HAL_DMA_GET_FLAG: gets the DMA Channels pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA Channels pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA Channels interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA Channels 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”).



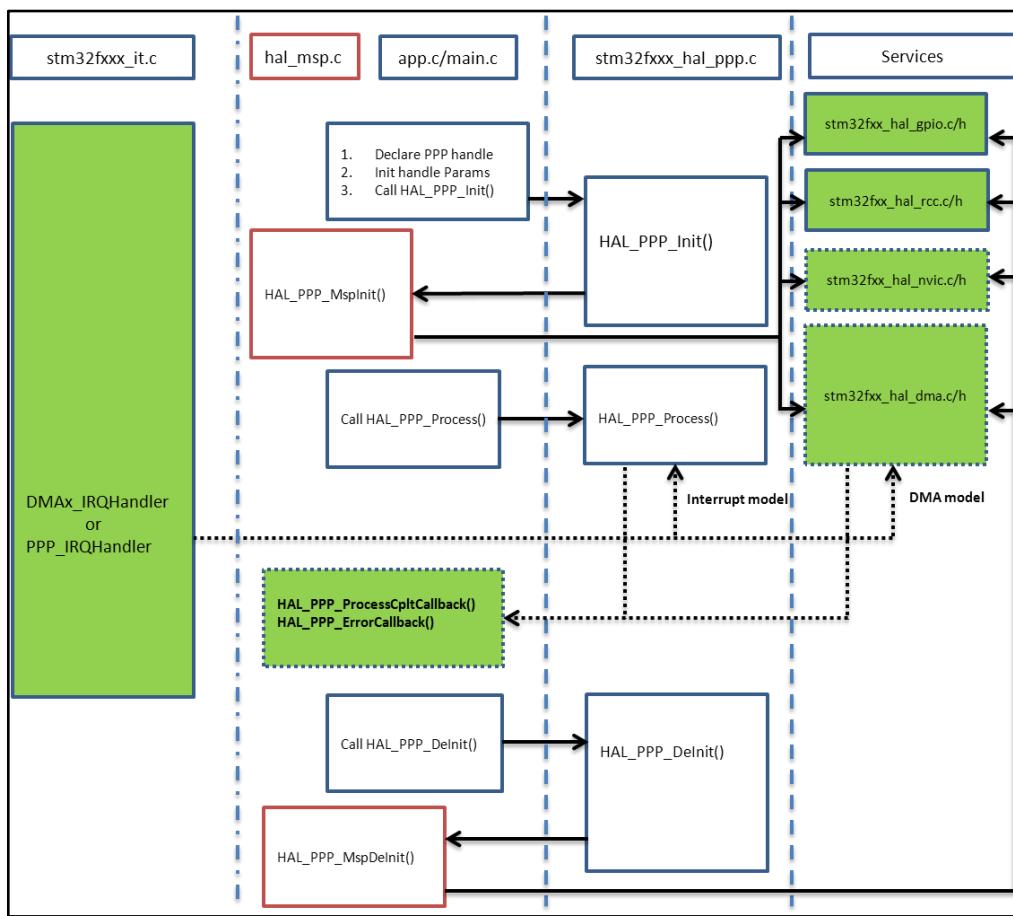
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

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 user application, 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 (see green blocks in the above schematics).

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 (see blue and red blocks in the above schematics).

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 `stm32l0xx_hal.c`.

- `HAL_Init()`: this function must be called at application startup to
 - Initialize data/instruction cache and pre-fetch queue
 - Set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
 - Call `HAL_MspInit()` user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). `HAL_MspInit()` is defined as “weak” empty function in the HAL drivers.

- HAL_DeInit()
 - Resets all peripherals
 - Calls function HAL_MspDeInit() which a is user callback function to do system level De-Initializations.
- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.

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

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in the application 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 MSI as source */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
RCC_OscInitStruct.MSIRANGE = RCC_MSIRANGE_5;
RCC_OscInitStruct.MSICalibrationValue=0x00;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
HAL_RCC_OscConfig(&RCC_OscInitStruct);
/* Select MSI 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_OscInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
RCC_OscInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_OscInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_OscInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0);
}
```

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_HandleTypeDefDef *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_HandleTypeDefDef *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 *stm32l0xx_hal_msp.c* file in the user folders. An *stm32l0xx_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.

stm32l0xx_hal_msp.c file contains the following functions:

Table 14: MSP functions

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

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

When one or more peripherals need 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 *stm32l0xx_it.c*.

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

An example of use is illustrated below:

main.c file:

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

stm32l0xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void USART1_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 *stm32l0xx_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_StatusTypeDef 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 = USART1;
HAL_UART_Init(&UartHandle);
(...)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(...)
    HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
    HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(...)
}

```

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

An example of use is illustrated below:

main.c file:

```

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

```

stm32l0xx_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 values are the following:

Table 15: Timeout values

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

Notes:

⁽¹⁾HAL_MAX_DELAY is defined in the stm32l0xx_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)
{
    uint32_t tickstart = 0;
    ...
    tickstart = HAL_GetTick();
    ...
    while(ProcessOngoing)
    {
        ...
        if((HAL_GetTick() - tickstart) > LOCAL_PROCESS_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)
{
    uint32_t tickstart = 0;
    ...
    tickstart = HAL_GetTick();
    ...
    while(ProcessOngoing)
    {
        ...
        if(Timeout != HAL_MAX_DELAY)
        {
```

```

if((HAL_GetTick() - tickstart ) > 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 */
PPP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR; /*return with HAL error */
```

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

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
    ErrorCode = HAL_PPP_GetError (hppp); /* 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 *stm32l0xx_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 *stm32l0xx_hal_conf.h*:

```
/* Exported macro -----*/
#ifndef 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 Generic Driver

3.1 HAL Firmware driver introduction

3.2 HAL Firmware driver API description

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

3.2.1 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

3.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

3.2.3 HAL Control functions

This section provides functions allowing to:



- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier
- Configures low power mode behavior when the MCU is in Debug mode
- `HAL_IncTick()`
- `HAL_GetTick()`
- `HAL_Delay()`
- `HAL_SuspendTick()`
- `HAL_ResumeTick()`
- `HAL_GetHalVersion()`
- `HAL_GetREVID()`
- `HAL_GetDEVID()`
- `HAL_DBG_LowPowerConfig()`
- `HAL_GetBootMode()`
- `HAL_I2CFastModePlusConfig()`
- `HAL_VREFINT_Cmd()`
- `HAL_VREFINT_OutputSelect()`
- `HAL_ADC_EnableBuffer_Cmd()`
- `HAL_ADC_EnableBufferSensor_Cmd()`
- `HAL_COMP_EnableBuffer_Cmd()`
- `HAL_RC48_EnableBuffer_Cmd()`
- `HAL_Lock_Cmd()`

3.2.3.1 HAL_Init

Function Name	HAL_StatusTypeDef HAL_Init (void)
Function Description	This function configures the Flash prefetch, Flash preread and Buffer cache, Configures time base source, NVIC and Low level hardware.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is called at the beginning of program after reset and before the clock configuration • The time base configuration is based on MSI clock when exiting from Reset. Once done, time base tick start incrementing. In the default implementation,Systick is used as source of time base. the tick variable is incremented each 1ms in its ISR.

3.2.3.2 HAL_DelInit

Function Name	HAL_StatusTypeDef HAL_DelInit (void)
Function Description	This function de-Initializes common part of the HAL and stops the source of time base.
Parameters	<ul style="list-style-type: none">None.
Return values	<ul style="list-style-type: none">HAL status
Notes	<ul style="list-style-type: none">This function is optional.

3.2.3.3 HAL_MspInit

Function Name	void HAL_MspInit (void)
Function Description	Initializes the MSP.
Parameters	<ul style="list-style-type: none">None.
Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

3.2.3.4 HAL_MspDeInit

Function Name	void HAL_MspDeInit (void)
Function Description	Deinitializes the MSP.
Parameters	<ul style="list-style-type: none">None.
Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

3.2.3.5 HAL_IncTick

Function Name	void HAL_IncTick (void)
Function Description	This function is called to increment a global variable "uwTick" used as application time base.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> In the default implementation, this variable is incremented each 1ms in Systick ISR. This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.2.3.6 HAL_GetTick

Function Name	uint32_t HAL_GetTick (void)
Function Description	Provides a tick value in millisecond.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> tick value
Notes	<ul style="list-style-type: none"> This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.2.3.7 HAL_Delay

Function Name	void HAL_Delay (__IO uint32_t Delay)
Function Description	This function provides accurate delay (in milliseconds) based on variable incremented.
Parameters	<ul style="list-style-type: none"> Delay : specifies the delay time length, in milliseconds.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented. This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.2.3.8 HAL_SuspendTick

Function Name	void HAL_SuspendTick (void)
Function Description	Suspend Tick increment.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the the SysTick interrupt will be disabled and so Tick increment is suspended. • This function is declared as __weak to be overwritten in case of other implementations in user file.

3.2.3.9 HAL_ResumeTick

Function Name	void HAL_ResumeTick (void)
Function Description	Resume Tick increment.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the the SysTick interrupt will be enabled and so Tick increment is resumed. • This function is declared as __weak to be overwritten in case of other implementations in user file.

3.2.3.10 HAL_GetHalVersion

Function Name	uint32_t HAL_GetHalVersion (void)
Function Description	Returns the HAL revision.

Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> version : 0xXYZR (8bits for each decimal, R for RC)
Notes	<ul style="list-style-type: none"> None.

3.2.3.11 HAL_GetREVID

Function Name	<code>uint32_t HAL_GetREVID (void)</code>
Function Description	Returns the device revision identifier.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> Device revision identifier
Notes	<ul style="list-style-type: none"> None.

3.2.3.12 HAL_GetDEVID

Function Name	<code>uint32_t HAL_GetDEVID (void)</code>
Function Description	Returns the device identifier.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> Device identifier
Notes	<ul style="list-style-type: none"> None.

3.2.3.13 HAL_DBG_LowPowerConfig

Function Name	<code>void HAL_DBG_LowPowerConfig (uint32_t Periph, FunctionalState NewState)</code>
Function Description	Configures low power mode behavior when the MCU is in Debug mode.
Parameters	<ul style="list-style-type: none"> Periph : specifies the low power mode. This parameter can

	be any combination of the following values:
	<ul style="list-style-type: none"> – DBGMCU_SLEEP : Keep debugger connection during SLEEP mode – DBGMCU_STOP : Keep debugger connection during STOP mode – DBGMCU_STANDBY : Keep debugger connection during STANDBY mode
• NewState :	new state of the specified low power mode in Debug mode. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

3.2.3.14 HAL_GetBootMode

Function Name	<code>uint32_t HAL_GetBootMode (void)</code>
Function Description	Returns the boot mode as configured by user.
Parameters	<ul style="list-style-type: none"> • None :
Return values	<ul style="list-style-type: none"> • The boot mode as configured by user. The returned value can be one of the following values: <ul style="list-style-type: none"> – <i>0x00000000: Boot is configured in Main Flash memory</i> – <i>0x00000100: Boot is configured in System Flash memory</i> – <i>0x00000300: Boot is configured in Embedded SRAM memory</i>
Notes	<ul style="list-style-type: none"> • None.

3.2.3.15 HAL_I2CFastModePlusConfig

Function Name	<code>void HAL_I2CFastModePlusConfig (uint32_t SYSCFG_I2CFastModePlus, FunctionalState NewState)</code>
Function Description	Configures the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none"> • SYSCFG_I2CFastModePlus : selects the pin. This parameter can be one of the following values: <ul style="list-style-type: none"> – SYSCFG_I2CFastModePlus_PB6 : Configure fast mode plus driving capability for PB6

	<ul style="list-style-type: none"> - <i>SYSCFG_I2CFastModePlus_PB7</i> : Configure fast mode plus driving capability for PB7 - <i>SYSCFG_I2CFastModePlus_PB8</i> : Configure fast mode plus driving capability for PB8 - <i>SYSCFG_I2CFastModePlus_PB9</i> : Configure fast mode plus driving capability for PB9 - <i>SYSCFG_I2CFastModePlus_I2C1</i> : Configure fast mode plus driving capability for I2C1 pins - <i>SYSCFG_I2CFastModePlus_I2C2</i> : Configure fast mode plus driving capability for I2C2 pins <ul style="list-style-type: none"> • NewState : This parameter can be: ENABLE: Enable fast mode plus driving capability for selected I2C pin DISABLE: Disable fast mode plus driving capability for selected I2C pin
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using SYSCFG_I2CFastModePlus_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9. • For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using SYSCFG_I2CFastModePlus_I2C1 parameter. • For all I2C2 pins fast mode plus driving capability can be enabled only by using SYSCFG_I2CFastModePlus_I2C2 parameter.

3.2.3.16 HAL_VREFINT_Cmd

Function Name	void HAL_VREFINT_Cmd (FunctionalState NewState)
Function Description	Enables or disables the VREFINT.
Parameters	<ul style="list-style-type: none"> • NewState : new state of the Vrefint. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

3.2.3.17 HAL_VREFINT_OutputSelect

Function Name	void HAL_VREFINT_OutputSelect (uint32_t SYSCFG_Vrefint_OUTPUT)
Function Description	Selects the output of internal reference voltage (VREFINT).
Parameters	<ul style="list-style-type: none"> • SYSCFG_Vrefint_OUTPUT : new state of the Vrefint output. This parameter can be one of the following values: <ul style="list-style-type: none"> – SYSCFG_VREFINT_OUT_NONE : – SYSCFG_VREFINT_OUT_PB0 : – SYSCFG_VREFINT_OUT_PB1 : – SYSCFG_VREFINT_OUT_PB0_PB1 :
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

3.2.3.18 HAL_ADC_EnableBuffer_Cmd

Function Name	void HAL_ADC_EnableBuffer_Cmd (FunctionalState NewState)
Function Description	Enables or disables the Buffer Vrefint for the ADC.
Parameters	<ul style="list-style-type: none"> • NewState : new state of the Vrefint. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • This is functional only if the LOCK is not set

3.2.3.19 HAL_ADC_EnableBufferSensor_Cmd

Function Name	void HAL_ADC_EnableBufferSensor_Cmd (FunctionalState NewState)
Function Description	Enables or disables the Buffer Sensor for the ADC.
Parameters	<ul style="list-style-type: none"> • NewState : new state of the Vrefint. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • This is functional only if the LOCK is not set.

3.2.3.20 HAL_COMP_EnableBuffer_Cmd

Function Name	void HAL_COMP_EnableBuffer_Cmd (FunctionalState NewState)
Function Description	Enables or disables the Buffer Vrefint for the COMP.
Parameters	<ul style="list-style-type: none"> • NewState : new state of the Vrefint. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • This is functional only if the LOCK is not set

3.2.3.21 HAL_RC48_EnableBuffer_Cmd

Function Name	void HAL_RC48_EnableBuffer_Cmd (FunctionalState NewState)
Function Description	Enables or disables the Buffer Vrefint for the RC48.
Parameters	<ul style="list-style-type: none"> • NewState : new state of the Vrefint. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • This is functional only if the LOCK is not set

3.2.3.22 HAL_Lock_Cmd

Function Name	void HAL_Lock_Cmd (FunctionalState NewState)
Function Description	Enables or disables the Lock.
Parameters	<ul style="list-style-type: none"> • NewState : new state of the Lock. This parameter can be: ENABLE or DISABLE.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

3.3 HAL Firmware driver defines

3.3.1 HAL

HAL

HAL_DBGMCU_Low_Power_Config

- #define: **DBGMCU_SLEEP** **DBGMCU_CR_DBG_SLEEP**

- #define: **DBGMCU_STOP** **DBGMCU_CR_DBG_STOP**

- #define: **DBGMCU_STANDBY** **DBGMCU_CR_DBG_STANDBY**

HAL_Exported_Constants

- #define: **__HAL_FREEZE_TIM2_DBGMCU** (**DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_TIM2_STOP)**)

- #define: **__HAL_FREEZE_TIM6_DBGMCU** (**DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_TIM6_STOP)**)

- #define: **__HAL_FREEZE_RTC_DBGMCU** (**DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_RTC_STOP)**)

- #define: **__HAL_FREEZE_WWDG_DBGMCU** (**DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_WWDG_STOP)**)

- #define: **__HAL_FREEZE_IWDG_DBGMCU** (**DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_IWDG_STOP)**)

- #define: `__HAL_FREEZE_I2C1_TIMEOUT_DBGMCU (DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_I2C1_STOP))`
- #define: `__HAL_FREEZE_I2C2_TIMEOUT_DBGMCU (DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_I2C2_STOP))`
- #define: `__HAL_FREEZE_LPTIMER_DBGMCU (DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_LPTIMER_STOP))`
- #define: `__HAL_FREEZE_TIM22_DBGMCU (DBGMCU->APB2FZ |= (DBGMCU_APB2_FZ_DBG_TIM22_STOP))`
- #define: `__HAL_FREEZE_TIM21_DBGMCU (DBGMCU->APB2FZ |= (DBGMCU_APB2_FZ_DBG_TIM21_STOP))`
- #define: `__HAL_UNFREEZE_TIM2_DBGMCU (DBGMCU->APB1FZ &= ~(DBGMCU_APB1_FZ_DBG_TIM2_STOP))`
- #define: `__HAL_UNFREEZE_TIM6_DBGMCU (DBGMCU->APB1FZ &= ~(DBGMCU_APB1_FZ_DBG_TIM6_STOP))`
- #define: `__HAL_UNFREEZE_RTC_DBGMCU (DBGMCU->APB1FZ &= ~(DBGMCU_APB1_FZ_DBG_RTC_STOP))`
- #define: `__HAL_UNFREEZE_WWDG_DBGMCU (DBGMCU->APB1FZ &= ~(DBGMCU_APB1_FZ_DBG_WWDG_STOP))`
- #define: `__HAL_UNFREEZE_IWDG_DBGMCU (DBGMCU->APB1FZ &= ~(DBGMCU_APB1_FZ_DBG_IWDG_STOP))`

- #define: ***__HAL_UNFREEZE_I2C1_TIMEOUT_DBGMCU (DBGMCU->APB1FZ &= ~(DBGMCU_APB1_FZ_DBG_I2C1_STOP))***

- #define: ***__HAL_UNFREEZE_I2C2_TIMEOUT_DBGMCU (DBGMCU->APB1FZ &= ~ (DBGMCU_APB1_FZ_DBG_I2C2_STOP))***

- #define: ***__HAL_UNFREEZE_LPTIMER_DBGMCU (DBGMCU->APB1FZ &= ~ (DBGMCU_APB1_FZ_DBG_LPTIMER_STOP))***

- #define: ***__HAL_UNFREEZE_TIM22_DBGMCU (DBGMCU->APB2FZ &= ~ (DBGMCU_APB2_FZ_DBG_TIM22_STOP))***

- #define: ***__HAL_UNFREEZE_TIM21_DBGMCU (DBGMCU->APB2FZ &= ~ (DBGMCU_APB2_FZ_DBG_TIM21_STOP))***

- #define: ***__HAL_REMAPMEMORY_FLASH (SYSCFG->CFG1 &= ~(SYSCFG_CFG1_MEM_MODE))***

- #define: ***__HAL_REMAPMEMORY_SYSTEMFLASH do {SYSCFG->MEMRMP &= ~(SYSCFG_CFG1_MEM_MODE); \ SYSCFG->MEMRMP |= SYSCFG_CFG1_MEM_MODE_0; \ }while(0);***

- #define: ***__HAL_REMAPMEMORY_SRAM do {SYSCFG->CFG1 &= ~(SYSCFG_CFG1_MEM_MODE); \ SYSCFG->CFG1 |= (SYSCFG_CFG1_MEM_MODE_0 | SYSCFG_CFG1_MEM_MODE_1); \ }while(0);***

HAL_SYSCFG_flags_definition

- #define: ***SYSCFG_FLAG_RC48 SYSCFG_CFGR3_REF_HSI48_RDYF***

- #define: ***SYSCFG_FLAG_SENSOR_ADC SYSCFG_CFGR3_SENSOR_ADC_RDYF***

- #define: **SYSCFG_FLAG_VREF_ADC SYSCFG_VREFINT_ADC_RDYF**

- #define: **SYSCFG_FLAG_VREF_COMP SYSCFG_CFGR3_VREFINT_COMP_RDYF**

- #define: **SYSCFG_FLAG_VREF_READY SYSCFG_CFGR3_VREFINT_RDYF**

HAL_SYSCFG_I2C_FastModePlus_Config

- #define: **SYSCFG_I2CFastModePlus_PB6 SYSCFG_CFGR2_I2C_PB6_FMP**

- #define: **SYSCFG_I2CFastModePlus_PB7 SYSCFG_CFGR2_I2C_PB7_FMP**

- #define: **SYSCFG_I2CFastModePlus_PB8 SYSCFG_CFGR2_I2C_PB8_FMP**

- #define: **SYSCFG_I2CFastModePlus_PB9 SYSCFG_CFGR2_I2C_PB9_FMP**

- #define: **SYSCFG_I2CFastModePlus_I2C1 SYSCFG_CFGR2_I2C1_FMP**
Enable Fast Mode Plus on I2C1 pins

- #define: **SYSCFG_I2CFastModePlus_I2C2 SYSCFG_CFGR2_I2C2_FMP**
Enable Fast Mode Plus on I2C2 pins

HAL_SYSCFG_VREFINT_OUT_SELECT

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

- #define: **SYSCFG_VREFINT_OUT_PB0 SYSCFG_CFGR3_VREF_OUT_0**

- #define: **SYSCFG_VREFINT_OUT_PB1 SYSCFG_CFGR3_VREF_OUT_1**
- #define: **SYSCFG_VREFINT_OUT_PB0_PB1 SYSCFG_CFGR3_VREF_OUT**

4 HAL ADC Generic Driver

4.1 ADC Firmware driver introduction

4.2 ADC Firmware driver registers structures

4.2.1 ADC_HandleTypeDef

ADC_HandleTypeDef is defined in the `stm32l0xx_hal_adc.h`

Data Fields

- **`ADC_TypeDef * Instance`**
- **`ADC_InitTypeDef Init`**
- **`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
- **`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

4.2.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the `stm32l0xx_hal_adc.h`

Data Fields

- **`uint32_t OversamplingMode`**
- **`ADC_OversamplingTypeDef Oversample`**
- **`uint32_t ClockPrescaler`**
- **`uint32_t Resolution`**
- **`uint32_t SamplingTime`**
- **`uint32_t ScanDirection`**

- `uint32_t DataAlign`
- `uint32_t ContinuousConvMode`
- `uint32_t DiscontinuousConvMode`
- `uint32_t ExternalTrigConvEdge`
- `uint32_t ExternalTrigConv`
- `uint32_t DMAContinuousRequests`
- `uint32_t EOCSelection`
- `uint32_t Overrun`
- `uint32_t LowPowerAutoWait`
- `uint32_t LowPowerFrequencyMode`
- `uint32_t LowPowerAutoOff`

Field Documentation

- **`uint32_t ADC_InitTypeDef::OversamplingMode`**
 - Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`ADC_OversamplingTypeDef ADC_InitTypeDef::Oversample`**
 - Specifies the Oversampling parameters. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::ClockPrescaler`**
 - Selects the ADC clock frequency. This parameter can be a value of [`ADC_ClockPrescaler`](#)
- **`uint32_t ADC_InitTypeDef::Resolution`**
 - Configures the ADC resolution mode. This parameter can be a value of [`ADC_Resolution`](#)
- **`uint32_t ADC_InitTypeDef::SamplingTime`**
 - The sample time value to be set for all channels. This parameter can be a value of [`ADC_sampling_times`](#)
- **`uint32_t ADC_InitTypeDef::ScanDirection`**
 - The scan sequence direction. This parameter can be a value of [`ADC_scan_direction`](#)
- **`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::ContinuousConvMode`**
 - Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::DiscontinuousConvMode`**
 - Specifies whether the conversion is performed in Complete-sequence/Discontinuous-sequence. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::ExternalTrigConvEdge`**
 - Select the external trigger edge and enable the trigger. This parameter can be a value of [`ADC_External_trigger_Edge`](#)
- **`uint32_t ADC_InitTypeDef::ExternalTrigConv`**
 - Select the external event used to trigger the start of conversion. This parameter can be a value of [`ADC_External_trigger_Source`](#)

- ***uint32_t ADC_InitTypeDef::DMAContinuousRequests***
 - Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer max pointer is reached. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32_t ADC_InitTypeDef::EOCSelection***
 - Specifies what EOC (End Of Conversion) flag is used for conversion polling and interruption: end of single channel conversion or end of channels conversions sequence. This parameter can be a value of [ADC_EOCSelection](#)
- ***uint32_t ADC_InitTypeDef::Overrun***
 - Select the behaviour in case of overrun: data preserved or overwritten. This parameter has an effect on regular channels only, including in DMA mode. This parameter can be a value of [ADC_Overrun](#)
- ***uint32_t ADC_InitTypeDef::LowPowerAutoWait***
 - Specifies the usage of dynamic low power Auto Delay: new conversion start only when the previous conversion (for regular channel) is completed. This avoids risk of overrun for low frequency application. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32_t ADC_InitTypeDef::LowPowerFrequencyMode***
 - When selecting an analog ADC clock frequency lower than 2.8MHz, it is mandatory to first enable the Low Frequency Mode. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- ***uint32_t ADC_InitTypeDef::LowPowerAutoOff***
 - When setting the AutoOff feature, the ADC is always powered off when not converting and automatically wakes-up when a conversion is started (by software or hardware trigger). This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.

4.2.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the `stm32l0xx_hal_adc.h`

Data Fields

- ***uint32_t Channel***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
 - the ADC channel to configure. This parameter can be a value of [ADC_channels](#)

4.2.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the `stm32l0xx_hal_adc.h`

Data Fields

- *uint32_t WatchdogMode*
- *uint32_t Channel*
- *uint32_t ITMode*
- *uint32_t HighThreshold*
- *uint32_t LowThreshold*

Field Documentation

- *uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode*
 - Configures the ADC analog watchdog mode: single/all channels. This parameter can be a value of [ADC_analog_watchdog_mode](#)
- *uint32_t ADC_AnalogWDGConfTypeDef::Channel*
 - Selects which ADC channel to monitor by analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel (parameter WatchdogMode) This parameter can be a value of [ADC_channels](#)
- *uint32_t ADC_AnalogWDGConfTypeDef::ITMode*
 - Specifies whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- *uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold*
 - Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.
- *uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold*
 - Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

4.2.5 ADC_Common_TypeDef

ADC_Common_TypeDef is defined in the stm32l051xx.h

Data Fields

- *__IO uint32_t CCR*

Field Documentation

- *__IO uint32_t ADC_Common_TypeDef::CCR*

4.2.6 ADC_TypeDef

ADC_TypeDef is defined in the stm32l051xx.h

Data Fields

- `__IO uint32_t ISR`
- `__IO uint32_t IER`
- `__IO uint32_t CR`
- `__IO uint32_t CFGR1`
- `__IO uint32_t CFGR2`
- `__IO uint32_t SMPR`
- `uint32_t RESERVED1`
- `uint32_t RESERVED2`
- `__IO uint32_t TR`
- `uint32_t RESERVED3`
- `__IO uint32_t CHSELR`
- `uint32_t RESERVED4`
- `__IO uint32_t DR`
- `uint32_t RESERVED5`
- `__IO uint32_t CALFACT`

Field Documentation

- `__IO uint32_t ADC_TypeDef::ISR`
 - ADC Interrupt and Status register, Address offset:0x00
- `__IO uint32_t ADC_TypeDef::IER`
 - ADC Interrupt Enable register, Address offset:0x04
- `__IO uint32_t ADC_TypeDef::CR`
 - ADC Control register, Address offset:0x08
- `__IO uint32_t ADC_TypeDef::CFG1`
 - ADC Configuration register 1, Address offset:0x0C
- `__IO uint32_t ADC_TypeDef::CFG2`
 - ADC Configuration register 2, Address offset:0x10
- `__IO uint32_t ADC_TypeDef::SMPR`
 - ADC Sampling time register, Address offset:0x14
- `uint32_t ADC_TypeDef::RESERVED1`
 - Reserved, 0x18
- `uint32_t ADC_TypeDef::RESERVED2`
 - Reserved, 0x1C
- `__IO uint32_t ADC_TypeDef::TR`
 - ADC watchdog threshold register, Address offset:0x20
- `uint32_t ADC_TypeDef::RESERVED3`
 - Reserved, 0x24
- `__IO uint32_t ADC_TypeDef::CHSELR`
 - ADC channel selection register, Address offset:0x28
- `uint32_t ADC_TypeDef::RESERVED4`
 - Reserved, 0x2C
- `__IO uint32_t ADC_TypeDef::DR`
 - ADC data register, Address offset:0x40
- `uint32_t ADC_TypeDef::RESERVED5`
 - Reserved, 0x44 - 0xB0
- `__IO uint32_t ADC_TypeDef::CALFACT`
 - ADC data register, Address offset:0xB4

4.3 ADC Firmware driver API description

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

4.3.1 ADC specific features

1. 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.
2. A built-in hardware oversampler allows to improve analog performances while off-loading the related computational burden from the CPU.
3. Interrupt generation at the end of conversion and in case of analog watchdog or overrun events.
4. Single and continuous conversion modes.
5. Scan or discontinuous mode conversion of channel 0 to channel 18.
6. Configurable scan direction (Upward from channel 0 to 18 or Backward from channel 18 to channel 0)
7. Data alignment with in-built data coherency.
8. Channel-wise programmable sampling time.
9. External trigger option with configurable polarity.
10. DMA request generation during regular channel conversion.
11. ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
12. ADC input range: VREF- =VIN =VREF+.
13. ADC self-calibration.
14. ADC is automatically powered off (AutoOff mode) except during the active conversion phase. This dramatically reduces the power consumption of the ADC.
15. Wait mode to prevent ADC overrun in applications with low frequency.

4.3.2 How to use this driver

1. Enable the ADC interface As prerequisite, into HAL_ADC_MspInit(), ADC clock must be configured at RCC top level. Depending on both possible clock sources: PCLK clock or ADC asynchronous clock. __ADC1_CLK_ENABLE();
2. 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();
3. Configure the ADC parameters (conversion resolution, oversampler, data alignment, continuous mode,...) using the HAL_ADC_Init() function.
4. Activate the ADC peripheral using one of the start functions: HAL_ADC_Start(), HAL_ADC_Start_IT() or HAL_ADC_Start_DMA()

Channels configuration

- To configure the ADC channels group, use HAL_ADC_ConfigChannel() function.
- To read the ADC converted values, use the HAL_ADC_GetValue() function.

DMA feature configuration

- To enable the DMA mode, use the HAL_ADC_Start_DMA() function.
- To enable the generation of DMA requests continuously at the end of the last DMA transfer, set .Init.DMAContinuousRequests to ENABLE and call HAL_ADC_Init() function.

4.3.3 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_DelInit\(\)*](#)
- [*HAL_ADC_MspInit\(\)*](#)
- [*HAL_ADC_MspDelInit\(\)*](#)

4.3.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- poll for conversion complete.
- poll for conversion event.
- Start conversion and enable interrupt.
- Stop conversion and disable interrupt.
- handle ADC interrupt request.
- Start conversion of regular channel and enable DMA transfer.
- Stop conversion of regular channel and disable DMA transfer.
- Get result of regular channel conversion.
- 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\(\)*](#)

4.3.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Configure the analog watch dog.
- [**HAL_ADC_ConfigChannel\(\)**](#)
- [**HAL_ADC_AnalogWDGConfig\(\)**](#)

4.3.6 ADC Peripheral State functions

This subsection provides functions allowing to

- Check the ADC state.
- handle ADC interrupt request.
- [**HAL_ADC_GetState\(\)**](#)
- [**HAL_ADC_GetError\(\)**](#)

4.3.6.1 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.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	HAL status
Notes	<ul style="list-style-type: none"> • 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).

4.3.6.2 HAL_ADC_DeInit

Function Name	HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)
Function Description	Deinitialize the ADC peripheral registers to its default reset values.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	HAL status

-
- | | |
|-------|--|
| Notes | <ul style="list-style-type: none"> • To not impact other ADCs, reset of common ADC registers have been left commented below. If needed, the example code can be copied and uncommented into function HAL_ADC_MspDeInit(). |
|-------|--|

4.3.6.3 HAL_ADC_MspInit

Function Name	void HAL_ADC_MspInit (<i>ADC_HandleTypeDef</i> * hadc)
Function Description	Initializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.4 HAL_ADC_MspDeInit

Function Name	void HAL_ADC_MspDeInit (<i>ADC_HandleTypeDef</i> * hadc)
Function Description	Deinitializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.5 HAL_ADC_Start

Function Name	HAL_StatusTypeDef HAL_ADC_Start (<i>ADC_HandleTypeDef</i> * hadc)
Function Description	Enables ADC and starts conversion of the regular channels.

Parameters	<ul style="list-style-type: none"> hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

4.3.6.6 HAL_ADC_Stop

Function Name	HAL_StatusTypeDef HAL_ADC_Stop (<i>ADC_HandleTypeDef * hadc</i>)
Function Description	Stop ADC conversion of regular channels, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

4.3.6.7 HAL_ADC_PollForConversion

Function Name	HAL_StatusTypeDef HAL_ADC_PollForConversion (<i>ADC_HandleTypeDef * hadc, uint32_t Timeout</i>)
Function Description	Poll for conversion complete.
Parameters	<ul style="list-style-type: none"> hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. Timeout : Timeout value in millisecond.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

4.3.6.8 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	<ul style="list-style-type: none"> • hadc : ADC handle. • EventType : the ADC event type. This parameter can be one of the following values: <ul style="list-style-type: none"> - AWD_EVENT : ADC Analog watchdog event. - OVR_EVENT : ADC Overrun event. • Timeout : Timeout value in millisecond.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

4.3.6.9 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	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • HAL status.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.10 HAL_ADC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular channels, disable interruptions EOC/EOS/OVR, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.11 HAL_ADC_IRQHandler

Function Name	void HAL_ADC_IRQHandler (<i>ADC_HandleTypeDef</i> * hadc)
Function Description	Handles ADC interrupt request.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.12 HAL_ADC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_ADC_Start_DMA (<i>ADC_HandleTypeDef</i> * hadc, uint32_t * pData, uint32_t Length)
Function Description	Enables ADC DMA request after last transfer (Single-ADC mode) and enables ADC peripheral.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.13 HAL_ADC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_ADC_Stop_DMA (<i>ADC_HandleTypeDef</i> * hadc)
---------------	--

Function Description	Disable ADC DMA (Single-ADC mode), disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.14 HAL_ADC_GetValue

Function Name	uint32_t HAL_ADC_GetValue (<i>ADC_HandleTypeDef</i> * hadc)
Function Description	Gets the converted value from data register of regular channel.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • Converted value
Notes	<ul style="list-style-type: none"> • None.

4.3.6.15 HAL_ADC_ConvCpltCallback

Function Name	void HAL_ADC_ConvCpltCallback (<i>ADC_HandleTypeDef</i> * hadc)
Function Description	Regular conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.16 HAL_ADC_ConvHalfCpltCallback

Function Name	void HAL_ADC_ConvHalfCpltCallback (<i>ADC_HandleTypeDef</i>* <i>hadc</i>)
Function Description	Regular conversion half DMA transfer callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.17 HAL_ADC_LevelOutOfWindowCallback

Function Name	void HAL_ADC_LevelOutOfWindowCallback (<i>ADC_HandleTypeDef</i>* <i>hadc</i>)
Function Description	Analog watchdog callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.18 HAL_ADC_ErrorCallback

Function Name	void HAL_ADC_ErrorCallback (<i>ADC_HandleTypeDef</i>* <i>hadc</i>)
Function Description	Error ADC callback.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

4.3.6.19 HAL_ADC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_ADC_ConfigChannel (<i>ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig</i>)
Function Description	Configures the selected ADC regular channel: sampling time, offset,.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • sConfig : ADC regular channel configuration structure.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

4.3.6.20 HAL_ADC_AnalogWDGConfig

Function Name	HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (<i>ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig</i>)
Function Description	Configures the analog watchdog.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

4.3.6.21 HAL_ADC_GetState

Function Name	HAL_ADC_StateTypeDef HAL_ADC_GetState (<i>ADC_HandleTypeDef * hadc</i>)
Function Description	return the ADC state
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • HAL state

-
- Notes • None.

4.3.6.22 HAL_ADC_GetError

Function Name	<code>uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)</code>
Function Description	Return the ADC error code.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • ADC Error Code
Notes	<ul style="list-style-type: none"> • None.

4.4 ADC Firmware driver defines

4.4.1 ADC

ADC

ADC_analog_watchdog_mode

- #define: `ADC_ANALOGWATCHDOG_NONE ((uint32_t) 0x00000000)`
- #define: `ADC_ANALOGWATCHDOG_SINGLE_REG ((uint32_t)(ADC_CFGR1_AWDSGL | ADC_CFGR1_AWDEN))`
- #define: `ADC_ANALOGWATCHDOG_ALL_REG ((uint32_t) ADC_CFGR1_AWDEN)`

ADC_channels

- #define: `ADC_CHANNEL_0 ((uint32_t)(ADC_CHSELR_CHSEL0))`

-
- #define: ***ADC_CHANNEL_1*** ((*uint32_t*)(***ADC_CHSELR_CHSEL1***) | ***ADC_CFGR1_AWDCH_0***)
 - #define: ***ADC_CHANNEL_2*** ((*uint32_t*)(***ADC_CHSELR_CHSEL2***) | ***ADC_CFGR1_AWDCH_1***)
 - #define: ***ADC_CHANNEL_3*** ((*uint32_t*)(***ADC_CHSELR_CHSEL3***) | ***ADC_CFGR1_AWDCH_1*** | ***ADC_CFGR1_AWDCH_0***)
 - #define: ***ADC_CHANNEL_4*** ((*uint32_t*)(***ADC_CHSELR_CHSEL4***) | ***ADC_CFGR1_AWDCH_2***)
 - #define: ***ADC_CHANNEL_5*** ((*uint32_t*)(***ADC_CHSELR_CHSEL5***) | ***ADC_CFGR1_AWDCH_2*** | ***ADC_CFGR1_AWDCH_0***)
 - #define: ***ADC_CHANNEL_6*** ((*uint32_t*)(***ADC_CHSELR_CHSEL6***) | ***ADC_CFGR1_AWDCH_2*** | ***ADC_CFGR1_AWDCH_1***)
 - #define: ***ADC_CHANNEL_7*** ((*uint32_t*)(***ADC_CHSELR_CHSEL7***) | ***ADC_CFGR1_AWDCH_2*** | ***ADC_CFGR1_AWDCH_1*** | ***ADC_CFGR1_AWDCH_0***)
 - #define: ***ADC_CHANNEL_8*** ((*uint32_t*)(***ADC_CHSELR_CHSEL8***) | ***ADC_CFGR1_AWDCH_3***)
 - #define: ***ADC_CHANNEL_9*** ((*uint32_t*)(***ADC_CHSELR_CHSEL9***) | ***ADC_CFGR1_AWDCH_3*** | ***ADC_CFGR1_AWDCH_0***)
 - #define: ***ADC_CHANNEL_10*** ((*uint32_t*)(***ADC_CHSELR_CHSEL10***) | ***ADC_CFGR1_AWDCH_3*** | ***ADC_CFGR1_AWDCH_1***)

- #define: **ADC_CHANNEL_11 ((uint32_t)(ADC_CHSEL_R_CHSEL11)|
ADC_CFGR1_AWDCH_3|ADC_CFGR1_AWDCH_1|ADC_CFGR1_AWDCH_0)**
- #define: **ADC_CHANNEL_12 ((uint32_t)(ADC_CHSEL_R_CHSEL12)|
ADC_CFGR1_AWDCH_3|ADC_CFGR1_AWDCH_2)**
- #define: **ADC_CHANNEL_13 ((uint32_t)(ADC_CHSEL_R_CHSEL13)|
ADC_CFGR1_AWDCH_3|ADC_CFGR1_AWDCH_2|ADC_CFGR1_AWDCH_0)**
- #define: **ADC_CHANNEL_14 ((uint32_t)(ADC_CHSEL_R_CHSEL14)|
ADC_CFGR1_AWDCH_3|ADC_CFGR1_AWDCH_2|ADC_CFGR1_AWDCH_1)**
- #define: **ADC_CHANNEL_15 ((uint32_t)(ADC_CHSEL_R_CHSEL15)|
ADC_CFGR1_AWDCH_3|ADC_CFGR1_AWDCH_2|ADC_CFGR1_AWDCH_1|
ADC_CFGR1_AWDCH_0)**
- #define: **ADC_CHANNEL_16 ((uint32_t)(ADC_CHSEL_R_CHSEL16)|
ADC_CFGR1_AWDCH_4)**
- #define: **ADC_CHANNEL_17 ((uint32_t)(ADC_CHSEL_R_CHSEL17)|
ADC_CFGR1_AWDCH_4|ADC_CFGR1_AWDCH_0)**
- #define: **ADC_CHANNEL_18 ((uint32_t)(ADC_CHSEL_R_CHSEL18)|
ADC_CFGR1_AWDCH_4|ADC_CFGR1_AWDCH_1)**
- #define: **ADC_CHANNEL_TEMPSENSOR ADC_CHANNEL_16**
- #define: **ADC_CHANNEL_VREFINT ADC_CHANNEL_17**

-
- #define: ***ADC_CHANNEL_VLCD ADC_CHANNEL_18***

ADC_Channel_AWD_Masks

- #define: ***ADC_CHANNEL_MASK ((uint32_t)0x0007FFFF)***

- #define: ***ADC_CHANNEL_AWD_MASK ((uint32_t)0x7C000000)***

ADC_ClockPrescaler

- #define: ***ADC_CLOCK_ASYNC_DIV1 ((uint32_t)0x00000000)***
ADC Asynchronous clock mode divided by 1

- #define: ***ADC_CLOCK_ASYNC_DIV2 (ADC_CCR_PRESC_0)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV4 (ADC_CCR_PRESC_1)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV6 (ADC_CCR_PRESC_1 | ADC_CCR_PRESC_0)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV8 (ADC_CCR_PRESC_2)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV10 (ADC_CCR_PRESC_2 | ADC_CCR_PRESC_0)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV12 (ADC_CCR_PRESC_2 | ADC_CCR_PRESC_1)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV16 (ADC_CCR_PRESC_2 | ADC_CCR_PRESC_1 | ADC_CCR_PRESC_0)***
ADC Asynchronous clock mode divided by 2

- #define: ***ADC_CLOCK_ASYNC_DIV32 (ADC_CCR_PRESC_3)***
ADC Asynchronous clock mode divided by 2

 - #define: ***ADC_CLOCK_ASYNC_DIV64 (ADC_CCR_PRESC_3 / ADC_CCR_PRESC_0)***
ADC Asynchronous clock mode divided by 2

 - #define: ***ADC_CLOCK_ASYNC_DIV128 (ADC_CCR_PRESC_3 / ADC_CCR_PRESC_1)***
ADC Asynchronous clock mode divided by 2

 - #define: ***ADC_CLOCK_ASYNC_DIV256 (ADC_CCR_PRESC_3 / ADC_CCR_PRESC_1 / ADC_CCR_PRESC_0)***
ADC Asynchronous clock mode divided by 2

 - #define: ***ADC_CLOCKPRESCALER_PCLK_DIV1 ((uint32_t)ADC_CFGR2_CKMODE_0)***
Synchronous clock mode divided by 1

 - #define: ***ADC_CLOCKPRESCALER_PCLK_DIV2 ((uint32_t)ADC_CFGR2_CKMODE_1)***
Synchronous clock mode divided by 2

 - #define: ***ADC_CLOCKPRESCALER_PCLK_DIV4 ((uint32_t)ADC_CFGR2_CKMODE)***
Synchronous clock mode divided by 4
- ADC_conversion_type***
- #define: ***REGULAR_GROUP ((uint32_t)(ADC_FLAG_EOC | ADC_FLAG_EOS))***

ADC_data_align

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

- #define: ***ADC_DATAALIGN_LEFT ((uint32_t)ADC_CFGR1_ALIGN)***

ADC_EOCSelection

- #define: ***EOC_SINGLE_CONV*** ((*uint32_t*) ***ADC_ISR_EOC***)
- #define: ***EOC_SEQ_CONV*** ((*uint32_t*) ***ADC_ISR_EOS***)
- #define: ***EOC_SINGLE_SEQ_CONV*** ((*uint32_t*)(***ADC_ISR_EOC*** | ***ADC_ISR_EOS***))
reserved for future use

ADC_Error_Code

- #define: ***HAL_ADC_ERROR_NONE*** ((*uint32_t*)0x00)
No error
- #define: ***HAL_ADC_ERROR_INTERNAL*** ((*uint32_t*)0x01)
ADC IP internal error: if problem of clocking, enable/disable, erroneous state
- #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

- #define: ***ADC_EXTERNALTRIG_EDGE_NONE*** ((*uint32_t*)0x00000000)
- #define: ***ADC_EXTERNALTRIG_EDGE_RISING***
((*uint32_t*)***ADC_CFGR1_EXTEN_0***)

- #define: **ADC_EXTERNALTRIG_EDGE_FALLING**
((*uint32_t*)ADC_CFGR1_EXTEN_1)
- #define: **ADC_EXTERNALTRIG_EDGE_RISINGFALLING**
((*uint32_t*)ADC_CFGR1_EXTEN)

ADC_External_trigger_Source

- #define: **ADC_EXTERNALTRIGO_T6_TRGO** ((*uint32_t*)0x00000000)
- #define: **ADC_EXTERNALTRIG1_T21_CC2 ADC_CFGR1_EXTSEL_0**
- #define: **ADC_EXTERNALTRIG2_T2_TRGO ADC_CFGR1_EXTSEL_1**
- #define: **ADC_EXTERNALTRIG3_T2_CC4** ((*uint32_t*)0x000000C0)
- #define: **ADC_EXTERNALTRIG4_T22_TRGO ADC_CFGR1_EXTSEL_2**
- #define: **ADC_EXTERNALTRIG7_EXT_IT11 ADC_CFGR1_EXTSEL**

ADC_flags_definition

- #define: **ADC_FLAG_RDY ADC_ISR_ADRDY**
ADC Ready (ADRDY) flag
- #define: **ADC_FLAG_EOSMP ADC_ISR_EOSMP**
ADC End of Sampling flag
- #define: **ADC_FLAG_EOC ADC_ISR_EOC**

ADC End of Regular Conversion flag

- #define: ***ADC_FLAG_EOS ADC_ISR_EOSEQ***

ADC End of Regular sequence of Conversions flag

- #define: ***ADC_FLAG_OVR ADC_ISR_OVR***

ADC overrun flag

- #define: ***ADC_FLAG_AWD ADC_ISR_AWD***

ADC Analog watchdog flag

- #define: ***ADC_FLAG_EOCAL ADC_ISR_EOCAL***

ADC End Of Calibration flag

- #define: ***ADC_FLAG_ALL (ADC_FLAG_RDY | ADC_FLAG_EOSMP | ADC_FLAG_EOC | ADC_FLAG_EOS | \ADC_FLAG_OVR | ADC_FLAG_AWD | ADC_FLAG_EOCAL)***

ADC_interrupts_definition

- #define: ***ADC_IT_RDY ADC_IER_ADRDYIE***

ADC Ready (ADRDY) interrupt source

- #define: ***ADC_IT_EOSMP ADC_IER_EOSMPIE***

ADC End of Sampling interrupt source

- #define: ***ADC_IT_EOC ADC_IER_EOCIE***

ADC End of Regular Conversion interrupt source

- #define: ***ADC_IT_EOS ADC_IER_EOSEQIE***

ADC End of Regular sequence of Conversions interrupt source

- #define: ***ADC_IT_OVR ADC_IER_OVRIE***

ADC overrun interrupt source

- #define: ***ADC_IT_AWD ADC_IER_AWDIE***

ADC Analog watchdog 1 interrupt source

- #define: **ADC_IT_EOCAL ADC_IER_EOCALIE**
ADC End of Calibration interrupt source

ADC_OVERRUN

- #define: **OVR_DATA_PRESERVED ((uint32_t)0x00000000)**
- #define: **OVR_DATA_OVERWRITTEN ((uint32_t)ADC_CFGR1_OVRMOD)**

ADC_Oversampling_Ratio

- #define: **ADC_OVERSAMPLING_RATIO_2 ((uint32_t)0x00000000)**
ADC Oversampling ratio 2x
- #define: **ADC_OVERSAMPLING_RATIO_4 ((uint32_t)0x00000004)**
ADC Oversampling ratio 4x
- #define: **ADC_OVERSAMPLING_RATIO_8 ((uint32_t)0x00000008)**
ADC Oversampling ratio 8x
- #define: **ADC_OVERSAMPLING_RATIO_16 ((uint32_t)0x0000000C)**
ADC Oversampling ratio 16x
- #define: **ADC_OVERSAMPLING_RATIO_32 ((uint32_t)0x00000010)**
ADC Oversampling ratio 32x
- #define: **ADC_OVERSAMPLING_RATIO_64 ((uint32_t)0x00000014)**
ADC Oversampling ratio 64x
- #define: **ADC_OVERSAMPLING_RATIO_128 ((uint32_t)0x00000018)**
ADC Oversampling ratio 128x
- #define: **ADC_OVERSAMPLING_RATIO_256 ((uint32_t)0x0000001C)**
ADC Oversampling ratio 256x

ADC_Resolution

- #define: ***ADC_RESOLUTION12b*** ((*uint32_t*)0x00000000)
ADC 12-bit resolution

- #define: ***ADC_RESOLUTION10b*** ((*uint32_t*)***ADC_CFGR1_RES_0***)
ADC 10-bit resolution

- #define: ***ADC_RESOLUTION8b*** ((*uint32_t*)***ADC_CFGR1_RES_1***)
ADC 8-bit resolution

- #define: ***ADC_RESOLUTION6b*** ((*uint32_t*)***ADC_CFGR1_RES***)
ADC 6-bit resolution

ADC_Right_Bit_Shift

- #define: ***ADC_RIGHTBITSHIFT_NONE*** ((*uint32_t*)0x00000000)
ADC No bit shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_1*** ((*uint32_t*)0x00000020)
ADC 1 bit shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_2*** ((*uint32_t*)0x00000040)
ADC 2 bits shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_3*** ((*uint32_t*)0x00000060)
ADC 3 bits shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_4*** ((*uint32_t*)0x00000080)
ADC 4 bits shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_5*** ((*uint32_t*)0x000000A0)
ADC 5 bits shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_6*** ((*uint32_t*)0x000000C0)
ADC 6 bits shift for oversampling

- #define: ***ADC_RIGHTBITSHIFT_7*** ((*uint32_t*)0x000000E0)

ADC 7 bits shift for oversampling

- #define: **ADC_RIGHTBITSHIFT_8** ((*uint32_t*)0x00000100)

ADC 8 bits shift for oversampling

ADC_sampling_times

- #define: **ADC_SAMPLETIME_1CYCLE_5** ((*uint32_t*)0x00000000)

ADC sampling time 1.5 cycle

- #define: **ADC_SAMPLETIME_7CYCLES_5** ((*uint32_t*)**ADC_SMPR_SMPR_0**)

ADC sampling time 7.5 CYCLES

- #define: **ADC_SAMPLETIME_13CYCLES_5** ((*uint32_t*)**ADC_SMPR_SMPR_1**)

ADC sampling time 13.5 CYCLES

- #define: **ADC_SAMPLETIME_28CYCLES_5** ((*uint32_t*)(**ADC_SMPR_SMPR_1** | **ADC_SMPR_SMPR_0**))

ADC sampling time 28.5 CYCLES

- #define: **ADC_SAMPLETIME_41CYCLES_5** ((*uint32_t*)**ADC_SMPR_SMPR_2**)

ADC sampling time 41.5 CYCLES

- #define: **ADC_SAMPLETIME_55CYCLES_5** ((*uint32_t*)(**ADC_SMPR_SMPR_2** | **ADC_SMPR_SMPR_0**))

ADC sampling time 55.5 CYCLES

- #define: **ADC_SAMPLETIME_71CYCLES_5** ((*uint32_t*)(**ADC_SMPR_SMPR_2** | **ADC_SMPR_SMPR_1**))

ADC sampling time 71.5 CYCLES

- #define: **ADC_SAMPLETIME_239CYCLES_5** ((*uint32_t*)**ADC_SMPR_SMPR**)

ADC sampling time 239.5 CYCLES

ADC_scan_direction

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

-
- #define: **ADC_SCAN_DIRECTION_BACKWARD ADC_CFGR1_SCANDIR**

ADC_TimeOut_Values

- #define: **ADC_ENABLE_TIMEOUT 10**
- #define: **ADC_DISABLE_TIMEOUT 10**
- #define: **ADC_STOP_CONVERSION_TIMEOUT 10**
- #define: **ADC_DELAY_10US_MIN_CPU_CYCLES 1800**

ADC_Triggered_Oversampling_Mode

- #define: **ADC_TRIGGEREDMODE_SINGLE_TRIGGER ((uint32_t)0x00000000)**
ADC No bit shift for oversampling
- #define: **ADC_TRIGGEREDMODE_MULTI_TRIGGER ((uint32_t)0x00000200)**
ADC No bit shift for oversampling

5 HAL ADC Extension Driver

5.1 ADCEx Firmware driver introduction

5.2 ADCEx Firmware driver API description

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

5.2.1 ADC specific features

1. Self calibration.

5.2.2 How to use this driver

1. Call HAL_ADCEx_Calibration_Start() to start calibration
2. Read the calibration factor using HAL_ADCEx_Calibration_GetValue()
3. User can set a his calibration factor using HAL_ADCEx_Calibration_SetValue()

5.2.3 ADC Extended features functions

This subsection provides functions allowing to:

- Start calibration.
- Get calibration factor.
- Set calibration factor.
- [**HAL_ADCEx_Calibration_Start\(\)**](#)
- [**HAL_ADCEx_Calibration_GetValue\(\)**](#)
- [**HAL_ADCEx_Calibration_SetValue\(\)**](#)

5.2.3.1 HAL_ADCEx_Calibration_Start

Function Name	HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)
Function Description	Start an automatic calibration.
Parameters	<ul style="list-style-type: none"> • hadc : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • SingleDiff : Selection of single-ended or differential input This parameter can be only of the following values: <ul style="list-style-type: none"> – ADC_SINGLE_ENDED : Channel in mode input single ended

Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

5.2.3.2 HAL_ADCEx_Calibration_GetValue

Function Name	<code>uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)</code>
Function Description	Get the calibration factor.
Parameters	<ul style="list-style-type: none"> • hadc : ADC handle. • SingleDiff : This parameter can be only: <ul style="list-style-type: none"> – ADC_SINGLE_ENDED : Channel in mode input single ended.
Return values	<ul style="list-style-type: none">• Calibration value.
Notes	<ul style="list-style-type: none">• None.

5.2.3.3 HAL_ADCEx_Calibration_SetValue

Function Name	<code>HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)</code>
Function Description	Set the calibration factor to overwrite automatic conversion result.
Parameters	<ul style="list-style-type: none"> • hadc : ADC handle • SingleDiff : This parameter can be only: <ul style="list-style-type: none"> – ADC_SINGLE_ENDED : Channel in mode input single ended. • CalibrationFactor : Calibration factor (coded on 7 bits maximum)
Return values	<ul style="list-style-type: none">• HAL state
Notes	<ul style="list-style-type: none">• None.

5.3 ADCEx Firmware driver defines

5.3.1 ADCEx

ADCEx

ADCEx_Channel_Mode

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

ADCEx_TimeOut_Values

- #define: ***ADC_CALIBRATION_TIMEOUT*** 10

6 HAL COMP Generic Driver

6.1 COMP Firmware driver introduction

6.2 COMP Firmware driver registers structures

6.2.1 COMP_HandleTypeDef

COMP_HandleTypeDef is defined in the `stm32l0xx_hal_comp.h`

Data Fields

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

Field Documentation

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

6.2.2 COMP_InitTypeDef

COMP_InitTypeDef is defined in the `stm32l0xx_hal_comp.h`

Data Fields

- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***uint32_t OutputPol***
- ***uint32_t Mode***
- ***uint32_t WindowMode***
- ***uint32_t TriggerMode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::InvertingInput***

- Selects the inverting input of the comparator. This parameter can be a value of ***COMP_InvertingInput***
- ***uint32_t COMP_InitTypeDef::NonInvertingInput***
 - Selects the non inverting input of the comparator. This parameter can be a value of ***COMP_NonInvertingInput***
- ***uint32_t COMP_InitTypeDef::OutputPol***
 - Selects the output polarity of the comparator. This parameter can be a value of ***COMP_OutputPolarity***
- ***uint32_t COMP_InitTypeDef::Mode***
 - Selects the operating consumption mode of the comparator to adjust the speed/consumption. This parameter can be a value of ***COMP_Mode***
- ***uint32_t COMP_InitTypeDef::WindowMode***
 - Selects the window mode of the comparator 2. This parameter can be a value of ***COMP_WindowMode***
- ***uint32_t COMP_InitTypeDef::TriggerMode***
 - Selects the trigger mode of the comparator (interrupt mode). This parameter can be a value of ***COMP_TriggerMode***

6.2.3 COMP_TypeDef

COMP_TypeDef is defined in the `stm32l051xx.h`

Data Fields

- ***__IO uint32_t CSR***

Field Documentation

- ***__IO uint32_t COMP_TypeDef::CSR***
 - COMP comparator control and status register, Address offset: 0x18

6.3 COMP Firmware driver API description

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

6.3.1.1 HAL_COMP_Init

Function Name	<i>HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)</i>
Function Description	Initializes the COMP according to the specified parameters in the <i>COMP_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • <i>hcomp</i> : COMP handle
Return values	<ul style="list-style-type: none"> • <i>HAL status</i>
Notes	<ul style="list-style-type: none"> • If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system

reset.

6.3.1.2 HAL_COMP_DeInit

Function Name	HAL_StatusTypeDef HAL_COMP_DeInit (<i>COMP_HandleTypeDef</i> * hcomp)
Function Description	Deinitializes the COMP peripheral.
Parameters	<ul style="list-style-type: none"> • hcomp : COMP handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

6.3.1.3 HAL_COMP_MspInit

Function Name	void HAL_COMP_MspInit (<i>COMP_HandleTypeDef</i> * hcomp)
Function Description	Initializes the COMP MSP.
Parameters	<ul style="list-style-type: none"> • hcomp : COMP handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

6.3.1.4 HAL_COMP_MspDeInit

Function Name	void HAL_COMP_MspDeInit (<i>COMP_HandleTypeDef</i> * hcomp)
Function Description	Deinitializes COMP MSP.
Parameters	<ul style="list-style-type: none"> • hcomp : COMP handle
Return values	<ul style="list-style-type: none"> • None.

Notes	<ul style="list-style-type: none">None.
-------	---

6.3.1.5 HAL_COMP_Start

Function Name	HAL_StatusTypeDef HAL_COMP_Start (<i>COMP_HandleTypeDef * hcomp</i>)
Function Description	Start the comparator.
Parameters	<ul style="list-style-type: none">hcomp : COMP handle
Return values	HAL status
Notes	<ul style="list-style-type: none">None.

6.3.1.6 HAL_COMP_Stop

Function Name	HAL_StatusTypeDef HAL_COMP_Stop (<i>COMP_HandleTypeDef * hcomp</i>)
Function Description	Stop the comparator.
Parameters	<ul style="list-style-type: none">hcomp : COMP handle
Return values	HAL status
Notes	<ul style="list-style-type: none">None.

6.3.1.7 HAL_COMP_Start_IT

Function Name	HAL_StatusTypeDef HAL_COMP_Start_IT (<i>COMP_HandleTypeDef * hcomp</i>)
Function Description	Enables the interrupt and starts the comparator.
Parameters	<ul style="list-style-type: none">hcomp : COMP handlemode : IT trigger mode: a value of COMP_TriggerMode

Return values	<ul style="list-style-type: none">• HAL status.
Notes	<ul style="list-style-type: none">• None.

6.3.1.8 HAL_COMP_Stop_IT

Function Name	HAL_StatusTypeDef HAL_COMP_Stop_IT (<i>COMP_HandleTypeDef * hcomp)</i>
Function Description	Disable the interrupt and Stop the comparator.
Parameters	<ul style="list-style-type: none">• hcomp : COMP handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

6.3.1.9 HAL_COMP_IRQHandler

Function Name	void HAL_COMP_IRQHandler (<i>COMP_HandleTypeDef * hcomp)</i>
Function Description	Comparator IRQ Handler.
Parameters	<ul style="list-style-type: none">• hcomp : COMP handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

6.3.1.10 HAL_COMP_Lock

Function Name	HAL_StatusTypeDef HAL_COMP_Lock (<i>COMP_HandleTypeDef * hcomp)</i>
Function Description	Lock the selected comparator configuration.
Parameters	<ul style="list-style-type: none">• hcomp : COMP handle

Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

6.3.1.11 HAL_COMP_GetOutputLevel

Function Name	<code>uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)</code>
Function Description	Return the output level (high or low) of the selected comparator.
Notes	<ul style="list-style-type: none">• None.

6.3.1.12 HAL_COMP_TriggerCallback

Function Name	<code>void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)</code>
Function Description	Comparator callback.
Parameters	<ul style="list-style-type: none">• hcomp : COMP handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

6.3.1.13 HAL_COMP_GetState

Function Name	<code>HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)</code>
Function Description	Return the COMP state.
Parameters	<ul style="list-style-type: none">• hcomp : : COMP handle
Return values	<ul style="list-style-type: none">• HAL state
Notes	<ul style="list-style-type: none">• None.

6.4 COMP Firmware driver defines

6.4.1 COMP

COMP

COMP_Exported_Constants

- #define: ***COMP_CSR_UPDATE_PARAMETERS_MASK*** ((*uint32_t*)0xC0008779)
- #define: ***COMP_LOCK_DISABLE*** ((*uint32_t*)0x00000000)
- #define: ***COMP_LOCK_ENABLE COMP_CSR_COMPxLOCK***
- #define: ***COMP_STATE_BIT_LOCK*** ((*uint32_t*)0x10)

COMP_ExtiLineEvent

- #define: ***COMP EXTI_LINE_COMP2_EVENT*** ((*uint32_t*)0x00400000)
External interrupt line 22 Connected to COMP2

- #define: ***COMP EXTI_LINE_COMP1_EVENT*** ((*uint32_t*)0x00200000)
External interrupt line 21 Connected to COMP1

COMP_InvertingInput

- #define: ***COMP_INVERTINGINPUT_VREFINT*** ((*uint32_t*)0x00000000)
VREFINT connected to comparator1 inverting input
- #define: ***COMP_INVERTINGINPUT_IO1*** ((*uint32_t*)0x00000010)
I/O1 connected to comparator inverting input (PA0) for COMP1 and (PA2) for COMP2
- #define: ***COMP_INVERTINGINPUT_DAC1*** ((*uint32_t*)0x00000020)
DAC1_OUT (PA4) connected to comparator inverting input

- #define: **COMP_INVERTINGINPUT_IO2** ((*uint32_t*)0x000000030)
I/O2 (PA5) connected to comparator inverting input
- #define: **COMP_INVERTINGINPUT_1_4VREFINT** ((*uint32_t*)0x000000040)
1/4 VREFINT connected to comparator inverting input
- #define: **COMP_INVERTINGINPUT_1_2VREFINT** ((*uint32_t*)0x000000050)
1/2 VREFINT connected to comparator inverting input
- #define: **COMP_INVERTINGINPUT_3_4VREFINT** ((*uint32_t*)0x000000060)
3/4 VREFINT connected to comparator inverting input
- #define: **COMP_INVERTINGINPUT_IO3** ((*uint32_t*)0x000000070)
I/O3 (PB3) for COMP2 connected to comparator inverting input

COMP_Mode

- #define: **COMP_MODE_HIGHSPEED** COMP_CSR_COMP2SPEED
High Speed
- #define: **COMP_MODE_LOWSPEED** ((*uint32_t*)0x00000000)
Low Speed

COMP_NonInvertingInput

- #define: **COMP_NONINVERTINGINPUT_IO1** ((*uint32_t*)0x00000000)
I/O1 (PA3) connected to comparator non inverting input
- #define: **COMP_NONINVERTINGINPUT_IO2** ((*uint32_t*)0x00000100)
I/O2 (PB4) connected to comparator non inverting input
- #define: **COMP_NONINVERTINGINPUT_IO3** ((*uint32_t*)0x00000200)
I/O3 (PB5) connected to comparator non inverting input
- #define: **COMP_NONINVERTINGINPUT_IO4** ((*uint32_t*)0x00000300)
I/O1 (PB6) connected to comparator non inverting input

- #define: **COMP_NONINVERTINGINPUT_IO5** ((*uint32_t*)0x00000400)
I/O3 (PB7) connected to comparator non inverting input

- #define: **COMP_NONINVERTINGINPUT_IO6** ((*uint32_t*)0x00000500)
I/O3 (PB7) connected to comparator non inverting input

- #define: **COMP_NONINVERTINGINPUT_IO7** ((*uint32_t*)0x00000600)
I/O3 (PB7) connected to comparator non inverting input

- #define: **COMP_NONINVERTINGINPUT_IO8** ((*uint32_t*)0x00000700)
I/O3 (PB7) connected to comparator non inverting input

COMP_OutputLevel

- #define: **COMP_OUTPUTLEVEL_LOW** ((*uint32_t*)0x00000000)
- #define: **COMP_OUTPUTLEVEL_HIGH COMP_CSR_COMPxOUTVALUE**

COMP_OutputPolarity

- #define: **COMP_OUTPUTPOL_NONINVERTED** ((*uint32_t*)0x00000000)
COMP output on GPIO isn't inverted
- #define: **COMP_OUTPUTPOL_INVERTED COMP_CSR_COMPxPOLARITY**
COMP output on GPIO is inverted

COMP_TriggerMode

- #define: **COMP_TRIGGERMODE_IT_RISING** ((*uint32_t*)0x00000001)
External Interrupt Mode with Rising edge trigger detection
- #define: **COMP_TRIGGERMODE_IT_FALLING** ((*uint32_t*)0x00000002)
External Interrupt Mode with Falling edge trigger detection
- #define: **COMP_TRIGGERMODE_IT_RISING_FALLING** ((*uint32_t*)0x00000003)
External Interrupt Mode with Rising/Falling edge trigger detection

COMP_WindowMode

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

Window mode disabled (Plus input of comparator 1 connected to PA1)

- #define: **COMP_WINDOWMODE_ENABLED COMP_CSR_COMP1WM**

Window mode enabled: Plus input of comparator 1 shorted with Plus input of comparator 2

7 HAL CORTEX Generic Driver

7.1 CORTEX Firmware driver introduction

7.2 CORTEX Firmware driver API description

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

7.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provide functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M0+ exceptions are managed by CMSIS functions.

1. Enable and Configure the priority of the selected IRQ Channels. The priority can be 0..3. Lower priority values gives higher priority. Priority Order: Lowest priority. Lowest hardware priority (IRQn position).
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority()
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ()

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base

- The HAL_SYSTICK_Config()function calls the SysTick_Config() function which is a CMSIS function that:
 - Configures the SysTick Reload register with value passed as function parameter.
 - Configures the SysTick IRQ priority to the lowest value (0x03).
 - 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 stm32l0xx_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 0xFFFFFFF

7.2.2 Initialization and de-initialization functions

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

- [*HAL_NVIC_SetPriority\(\)*](#)
- [*HAL_NVIC_EnableIRQ\(\)*](#)
- [*HAL_NVIC_DisableIRQ\(\)*](#)
- [*HAL_NVIC_SystemReset\(\)*](#)
- [*HAL_SYSTICK_Config\(\)*](#)

7.2.3 Peripheral Control functions

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

- [*HAL_NVIC_SetPendingIRQ\(\)*](#)
- [*HAL_NVIC_GetPendingIRQ\(\)*](#)
- [*HAL_NVIC_ClearPendingIRQ\(\)*](#)
- [*HAL_SYSTICK_CLKSourceConfig\(\)*](#)
- [*HAL_SYSTICK_IRQHandler\(\)*](#)
- [*HAL_SYSTICK_Callback\(\)*](#)

7.2.3.1 HAL_NVIC_SetPriority

Function Name	<code>void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)</code>
Function Description	Sets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> • 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 stm32l0xx.h file) • PreemptPriority : The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 3. A lower priority value indicates a higher priority • SubPriority : The subpriority level for the IRQ channel. with stm32l0xx devices, this parameter is a dummy value and it is ignored, because no subpriority supported in Cortex M0+ based products.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

7.2.3.2 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	<ul style="list-style-type: none"> 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 stm32l0xx.h file)
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

7.2.3.3 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	<ul style="list-style-type: none"> 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 stm32l0xx.h file)
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

7.2.3.4 HAL_NVIC_SystemReset

Function Name	void HAL_NVIC_SystemReset (void)
Function Description	Initiates a system reset request to reset the MCU.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

7.2.3.5 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	<ul style="list-style-type: none"> • TicksNumb : Specifies the ticks Number of ticks between two interrupts.
Return values	<ul style="list-style-type: none"> • status : - 0 Function succeeded. <ul style="list-style-type: none"> – 1 Function failed.
Notes	<ul style="list-style-type: none"> • None.

7.2.3.6 HAL_NVIC_SetPendingIRQ

Function Name	void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)
Function Description	Sets Pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> • 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 stm32l0xx.h file)
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

7.2.3.7 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	<ul style="list-style-type: none"> • 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 stm32l0xx.h file)

Return values	<ul style="list-style-type: none"> status : - 0 Interrupt status is not pending. - 1 Interrupt status is pending.
Notes	<ul style="list-style-type: none"> None.

7.2.3.8 HAL_NVIC_ClearPendingIRQ

Function Name	void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)
Function Description	Clears the pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> 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 stm32l0xx.h file)
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

7.2.3.9 HAL_SYSTICK_CLKSourceConfig

Function Name	void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)
Function Description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none"> CLKSource : specifies the SysTick clock source. This parameter can be one of the following values: <ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

7.2.3.10 HAL_SYSTICK_IRQHandler

Function Name	void HAL_SYSTICK_IRQHandler (void)
Function Description	This function handles SYSTICK interrupt request.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

7.2.3.11 HAL_SYSTICK_Callback

Function Name	void HAL_SYSTICK_Callback (void)
Function Description	SYSTICK callback.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

7.3 CORTEX Firmware driver defines

7.3.1 CORTEX

CORTEX

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 introduction

8.2 CRC Firmware driver registers structures

8.2.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the `stm32l0xx_hal_crc.h`

Data Fields

- `uint8_t DefaultPolynomialUse`
- `uint8_t DefaultInitValueUse`
- `uint32_t GeneratingPolynomial`
- `uint32_t CRCLength`
- `uint32_t InitValue`
- `uint32_t InputDataInversionMode`
- `uint32_t OutputDataInversionMode`

Field Documentation

- `uint8_t CRC_InitTypeDef::DefaultPolynomialUse`
 - This parameter is a value of [CRC_Default_Polynomial](#)
- `uint8_t CRC_InitTypeDef::DefaultInitValueUse`
 - This parameter is a value of [CRC_Default_InitValue_Use](#)
- `uint32_t CRC_InitTypeDef::GeneratingPolynomial`
 - Set CRC generating polynomial. 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal representation, e.g., for a polynomial of degree 7, $X^7 + X^6 + X^5 + X^2 + 1$ is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE
- `uint32_t CRC_InitTypeDef::CRCLength`
 - This parameter is a value of [CRC_Polynomial_Size_Definitions](#)
- `uint32_t CRC_InitTypeDef::InitValue`
 - Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE
- `uint32_t CRC_InitTypeDef::InputDataInversionMode`
 - This parameter is a value of [CRC_Input_Data_Inversion](#)
- `uint32_t CRC_InitTypeDef::OutputDataInversionMode`
 - This parameter is a value of [CRC_Output_Data_Inversion](#)

8.2.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the `stm32l0xx_hal_crc.h`

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
 - Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***
 - CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
 - CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
 - CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***
 - This parameter is a value of [CRC_Input_Buffer_Format](#)

8.2.3 CRC_TypeDef

CRC_TypeDef is defined in the stm32l051xx.h

Data Fields

- ***__IO uint32_t DR***
- ***__IO uint8_t IDR***
- ***uint8_t RESERVED0***
- ***uint16_t RESERVED1***
- ***__IO uint32_t CR***
- ***uint32_t RESERVED2***
- ***__IO uint32_t INIT***
- ***__IO uint32_t POL***

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

- `uint32_t CRC_TypeDef::RESERVED2`
 - Reserved, 0x0C
- `_IO uint32_t CRC_TypeDef::INIT`
 - Initial CRC value register, Address offset: 0x10
- `_IO uint32_t CRC_TypeDef::POL`
 - CRC polynomial register, Address offset: 0x14

8.3 CRC Firmware driver API description

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

8.3.1 Initialization and de-initialization functions

This section provides functions allowing to:

1. Initialize the CRC according to the specified parameters in the `CRC_InitTypeDef` and create the associated handle
 2. Deinitialize the CRC peripheral
 3. Initialize the CRC MSP
 4. Deinitialize CRC MSP
- `HAL_CRC_Init()`
 - `HAL_CRC_DelInit()`
 - `HAL_CRC_MspInit()`
 - `HAL_CRC_MspDelInit()`
 - `HAL_CRC_Input_Data_Reverse()`
 - `HAL_CRC_Output_Data_Reverse()`

8.3.2 Peripheral Control functions

This section provides functions allowing to:

1. Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using combination of the previous CRC value and the new one. or
 2. Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.
- `HAL_CRC_Accumulate()`
 - `HAL_CRC_Calculate()`

8.3.3 Peripheral State functions

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

- `HAL_CRC_GetState()`

8.3.3.1 HAL_CRC_Init

Function Name	<code>HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)</code>
---------------	--

Function Description	Initializes the CRC according to the specified parameters in the <code>CRC_InitTypeDef</code> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

8.3.3.2 HAL_CRC_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)</code>
Function Description	Deinitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

8.3.3.3 HAL_CRC_MspInit

Function Name	<code>void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)</code>
Function Description	Initializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

8.3.3.4 HAL_CRC_MspDeInit

Function Name	<code>void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)</code>
Function Description	Deinitializes the CRC MSP.

Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

8.3.3.5 HAL_CRC_Input_Data_Reverse

Function Name	HAL_StatusTypeDef HAL_CRC_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)
Function Description	Set the Reverse Input data mode.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle • InputReverseMode : Input Data inversion mode This parameter can be one of the following values: <ul style="list-style-type: none"> – CRC_INPUTDATA_INVERSION_NONE : no change in bit order (default value) – CRC_INPUTDATA_INVERSION_BYTE : Byte-wise bit reversal – CRC_INPUTDATA_INVERSION_HALFWORD : HalfWord-wise bit reversal – CRC_INPUTDATA_INVERSION_WORD : Word-wise bit reversal
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

8.3.3.6 HAL_CRC_Output_Data_Reverse

Function Name	HAL_StatusTypeDef HAL_CRC_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)
Function Description	Set the Reverse Output data mode.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle • OutputReverseMode : Output Data inversion mode This parameter can be one of the following values: <ul style="list-style-type: none"> – CRC_OUTPUTDATA_INVERSION_DISABLED : no CRC inversion (default value) – CRC_OUTPUTDATA_INVERSION_ENABLED : bit-level inversion (e.g for a 8-bit CRC: 0xB5 becomes 0xAD)

Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

8.3.3.7 HAL_CRC_Accumulate

Function Name	<code>uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)</code>
Function Description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle • pBuffer : pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • BufferLength : input data buffer length
Return values	<ul style="list-style-type: none"> • uint32_t CRC (returned value LSBs for CRC shorter than 32 bits)
Notes	<ul style="list-style-type: none">• None.

8.3.3.8 HAL_CRC_Calculate

Function Name	<code>uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)</code>
Function Description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle • pBuffer : pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • BufferLength : input data buffer length
Return values	<ul style="list-style-type: none"> • uint32_t CRC (returned value LSBs for CRC shorter than 32 bits)
Notes	<ul style="list-style-type: none">• None.

8.3.3.9 HAL_CRC_GetState

Function Name	<code>HAL_CRC_StateTypeDef HAL_CRC_GetState (</code> <code>CRC_HandleTypeDef * hcrc)</code>
Function Description	Returns the CRC state.
Parameters	<ul style="list-style-type: none"> • <code>hcrc</code> : CRC handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

8.4 CRC Firmware driver defines

8.4.1 CRC

CRC

CRC_Default_InitValue_Use

- #define: `DEFAULT_INIT_VALUE_ENABLE ((uint8_t)0x00)`
- #define: `DEFAULT_INIT_VALUE_DISABLE ((uint8_t)0x01)`

CRC_Default_Init_Value

- #define: `DEFAULT_CRC_INITVALUE 0xFFFFFFFF`

CRC_Default_Polynomial

- #define: `DEFAULT_POLYNOMIAL_ENABLE ((uint8_t)0x00)`
- #define: `DEFAULT_POLYNOMIAL_DISABLE ((uint8_t)0x01)`

CRC_Default_Polynomial_Value

-
- #define: **DEFAULT_CRC32_POLY** 0x04C11DB7

CRC_Input_Buffer_Format

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

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

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

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

CRC_Input_Data_Inversion

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

- #define: **CRC_INPUTDATA_INVERSION_BYTE** ((uint32_t)CRC_CR_REV_IN_0)

- #define: **CRC_INPUTDATA_INVERSION_HALFWORD** ((uint32_t)CRC_CR_REV_IN_1)

- #define: **CRC_INPUTDATA_INVERSION_WORD** ((uint32_t)CRC_CR_REV_IN)

CRC_Output_Data_Inversion

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

- #define: **CRC_OUTPUTDATA_INVERSION_ENABLED** ((uint32_t)CRC_CR_REV_OUT)

CRC_Polynomial_Sizes

- #define: ***CRC_POLYLENGTH_32B*** ((*uint32_t*)0x00000000)
- #define: ***CRC_POLYLENGTH_16B*** ((*uint32_t*)***CRC_CR_POLYSIZE_0***)
- #define: ***CRC_POLYLENGTH_8B*** ((*uint32_t*)***CRC_CR_POLYSIZE_1***)
- #define: ***CRC_POLYLENGTH_7B*** ((*uint32_t*)***CRC_CR_POLYSIZE***)

CRC_Polynomial_Size_Definitions

- #define: ***HAL_CRC_LENGTH_32B*** 32
- #define: ***HAL_CRC_LENGTH_16B*** 16
- #define: ***HAL_CRC_LENGTH_8B*** 8
- #define: ***HAL_CRC_LENGTH_7B*** 7

9 HAL CRC Extension Driver

9.1 CRCEEx Firmware driver introduction

9.2 CRCEEx Firmware driver API description

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

9.2.1 CRC Extended features functions

This subsection provides function allowing to:

- Set CRC polynomial if different from default one.
- [**HAL_CRCEEx_Polynomial_Set\(\)**](#)

9.2.1.1 [**HAL_CRCEEx_Polynomial_Set**](#)

Function Name	<code>HAL_StatusTypeDef HAL_CRCEEx_Polynomial_Set (</code> CRC_HandleTypeDef <code>* hcrc, uint32_t Pol, uint32_t PolyLength)</code>
Function Description	Initializes the CRC polynomial if different from default one.
Parameters	<ul style="list-style-type: none"> • hcrc : CRC handle • Pol : CRC generating polynomial (7, 8, 16 or 32-bit long) This parameter is written in normal representation, e.g. for a polynomial of degree 7, $X^7 + X^6 + X^5 + X^2 + 1$ is written 0x65 for a polynomial of degree 16, $X^{16} + X^{12} + X^5 + 1$ is written 0x1021 • PolyLength : CRC polynomial length This parameter can be one of the following values: <ul style="list-style-type: none"> - CRC_POLYLENGTH_7B : 7-bit long CRC (generating polynomial of degree 7) - CRC_POLYLENGTH_8B : 8-bit long CRC (generating polynomial of degree 8) - CRC_POLYLENGTH_16B : 16-bit long CRC (generating polynomial of degree 16) - CRC_POLYLENGTH_32B : 32-bit long CRC (generating polynomial of degree 32)
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

9.3 CRCEEx Firmware driver defines

9.3.1 CRCEEx

CRCEEx

10 HAL CRYP Generic Driver

10.1 CRYP Firmware driver introduction

10.2 CRYP Firmware driver registers structures

10.2.1 CRYP_HandleTypeDef

CRYP_HandleTypeDef is defined in the `stm32l0xx_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

10.2.2 CRYP_InitTypeDef

CRYP_InitTypeDef is defined in the stm32l0xx_hal_cryp.h

Data Fields

- *uint32_t DataType*
- *uint8_t * pKey*
- *uint8_t * pInitVect*

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**
- *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

10.2.3 AES_TypeDef

AES_TypeDef is defined in the stm32l061xx.h

Data Fields

- *__IO uint32_t CR*
- *__IO uint32_t SR*
- *__IO uint32_t DINR*
- *__IO uint32_t DOUTR*
- *__IO uint32_t KEYR0*
- *__IO uint32_t KEYR1*
- *__IO uint32_t KEYR2*
- *__IO uint32_t KEYR3*
- *__IO uint32_t IVR0*
- *__IO uint32_t IVR1*
- *__IO uint32_t IVR2*
- *__IO uint32_t IVR3*

Field Documentation

- *__IO uint32_t AES_TypeDef::CR*
 - AES control register, Address offset: 0x00
- *__IO uint32_t AES_TypeDef::SR*

- AES status register, Address offset: 0x04
- `__IO uint32_t AES_TypeDef::DINR`
- AES data input register, Address offset: 0x08
- `__IO uint32_t AES_TypeDef::DOUTR`
- AES data output register, Address offset: 0x0C
- `__IO uint32_t AES_TypeDef::KEYR0`
- AES key register 0, Address offset: 0x10
- `__IO uint32_t AES_TypeDef::KEYR1`
- AES key register 1, Address offset: 0x14
- `__IO uint32_t AES_TypeDef::KEYR2`
- AES key register 2, Address offset: 0x18
- `__IO uint32_t AES_TypeDef::KEYR3`
- AES key register 3, Address offset: 0x1C
- `__IO uint32_t AES_TypeDef::IVR0`
- AES initialization vector register 0, Address offset: 0x20
- `__IO uint32_t AES_TypeDef::IVR1`
- AES initialization vector register 1, Address offset: 0x24
- `__IO uint32_t AES_TypeDef::IVR2`
- AES initialization vector register 2, Address offset: 0x28
- `__IO uint32_t AES_TypeDef::IVR3`
- AES initialization vector register 3, Address offset: 0x2C

10.3 CRYP Firmware driver API description

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

10.3.1 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_DelInit\(\)`](#)
- [`HAL_CRYP_MspInit\(\)`](#)
- [`HAL_CRYP_MspDelInit\(\)`](#)

10.3.2 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES algorithm in different chaining modes
- Decrypt cypher text using AES algorithm in different 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\(\)`](#)

10.3.3 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error
- [`HAL_CRYP_ComputationCpltCallback\(\)`](#)
- [`HAL_CRYP_ErrorCallback\(\)`](#)
- [`HAL_CRYP_InCpltCallback\(\)`](#)
- [`HAL_CRYP_OutCpltCallback\(\)`](#)

10.3.4 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

- [`HAL_CRYP_IRQHandler\(\)`](#)

10.3.5 Peripheral State functions

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

- [`HAL_CRYP_GetState\(\)`](#)

10.3.5.1 HAL_CRYP_Init

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)</code>
Function Description	Initializes the CRYP according to the specified parameters in the

CRYP_InitTypeDef and creates the associated handle.

Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.2 HAL_CRYP_DelInit

Function Name	HAL_StatusTypeDef HAL_CRYP_DelInit (CRYP_HandleTypeDef * hcryp)
Function Description	Deinitializes the CRYP peripheral.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.3 HAL_CRYP_MspInit

Function Name	void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)
Function Description	Initializes the CRYP MSP.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

10.3.5.4 HAL_CRYP_MspDelInit

Function Name	void HAL_CRYP_MspDelinit (<i>CRYP_HandleTypeDef</i> * hcryp)
Function Description	Deinitializes CRYP MSP.
Parameters	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

10.3.5.5 HAL_CRYP_AESECB_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt (<i>CRYP_HandleTypeDef</i> * 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	<ul style="list-style-type: none"> • 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 ciphertext buffer • Timeout : Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.6 HAL_CRYP_AESCBC_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt (<i>CRYP_HandleTypeDef</i> * 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	<ul style="list-style-type: none"> • 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 ciphertext buffer

-
- | | |
|---------------|--|
| Return values | <ul style="list-style-type: none"> • Timeout : Specify Timeout value |
| Notes | <ul style="list-style-type: none"> • HAL status • None. |

10.3.5.7 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	<ul style="list-style-type: none"> • 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 ciphertext buffer • Timeout : Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.8 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	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData : Pointer to the ciphertext 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.9 HAL_CRYP_AESCBC_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code> <code>uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData : Pointer to the ciphertext 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	<ul style="list-style-type: none"> • None.

10.3.5.10 HAL_CRYP_AESCTR_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code> <code>uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData : Pointer to the ciphertext 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	<ul style="list-style-type: none"> • None.

10.3.5.11 HAL_CRYP_AESECB_Encrypt_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> • 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 ciphertext buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.12 HAL_CRYP_AESCBC_Encrypt_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> • 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 ciphertext buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.13 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	<ul style="list-style-type: none"> • 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 ciphertext buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.14 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	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData : Pointer to the ciphertext buffer • Size : Length of the plaintext buffer, must be a multiple of 16. • pPlainData : Pointer to the plaintext buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.15 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	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module pCypherData : Pointer to the ciphertext buffer Size : Length of the plaintext buffer, must be a multiple of 16 pPlainData : Pointer to the plaintext buffer
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.16 HAL_CRYP_AESCTR_Decrypt_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</code>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module pCypherData : Pointer to the ciphertext buffer Size : Length of the plaintext buffer, must be a multiple of 16 pPlainData : Pointer to the plaintext buffer
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.17 HAL_CRYP_AESECB_Encrypt_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> 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

Return values	<ul style="list-style-type: none"> pCypherData : Pointer to the ciphertext buffer HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.18 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	<ul style="list-style-type: none"> 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 ciphertext buffer
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.19 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	<ul style="list-style-type: none"> 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 ciphertext buffer
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

10.3.5.20 HAL_CRYP_AESECB_Decrypt_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</code>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData : Pointer to the ciphertext buffer • Size : Length of the plaintext buffer, must be a multiple of 16 bytes • pPlainData : Pointer to the plaintext buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.21 HAL_CRYP_AESCBC_Decrypt_DMA

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

10.3.5.22 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	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData : Pointer to the ciphertext buffer • Size : Length of the plaintext buffer, must be a multiple of 16 • pPlainData : Pointer to the plaintext buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

10.3.5.23 HAL_CRYP_ComputationCpltCallback

Function Name	void HAL_CRYP_ComputationCpltCallback (CRYP_HandleTypeDef * hcryp)
Function Description	Input Computation completed callbacks.
Parameters	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

10.3.5.24 HAL_CRYP_ErrorCallback

Function Name	void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)
Function Description	CRYP error callbacks.
Parameters	<ul style="list-style-type: none"> • hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

10.3.5.25 HAL_CRYP_InCpltCallback

Function Name	void HAL_CRYP_InCpltCallback (<i>CRYP_HandleTypeDef</i> * hcryp)
Function Description	Input transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

10.3.5.26 HAL_CRYP_OutCpltCallback

Function Name	void HAL_CRYP_OutCpltCallback (<i>CRYP_HandleTypeDef</i> * hcryp)
Function Description	Output transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

10.3.5.27 HAL_CRYP_IRQHandler

Function Name	void HAL_CRYP_IRQHandler (<i>CRYP_HandleTypeDef</i> * hcryp)
Function Description	This function handles CRYP interrupt request.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that

	contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

10.3.5.28 HAL_CRYP_GetState

Function Name	HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)
Function Description	Returns the CRYP state.
Parameters	<ul style="list-style-type: none"> hcryp : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> HAL state
Notes	<ul style="list-style-type: none"> None.

10.4 CRYP Firmware driver defines

10.4.1 CRYP

CRYP

CRYP_AlgoModeDirection

- #define: ***CRYP_CR_ALGOMODE_DIRECTION***
(uint32_t)(AES_CR_MODE|AES_CR_CHMOD)
- #define: ***CRYP_CR_ALGOMODE_AES_ECB_ENCRYPT*** ((uint32_t)0x00000000)
- #define: ***CRYP_CR_ALGOMODE_AES_ECB_KEYDERDECRYPT***
(AES_CR_MODE)
- #define: ***CRYP_CR_ALGOMODE_AES_CBC_ENCRYPT*** (AES_CR_CHMOD_0)

- #define: ***CRYP_CR_ALGOMODE_AES_CBC_KEYDERDECRYPT***
((*uint32_t*)(AES_CR_CHMOD_0|AES_CR_MODE))
 - #define: ***CRYP_CR_ALGOMODE_AES_CTR_ENCRYPT (AES_CR_CHMOD_1)***
 - #define: ***CRYP_CR_ALGOMODE_AES_CTR_DECRYPT***
((*uint32_t*)(AES_CR_CHMOD_1 | AES_CR_MODE_1))
-
- CRYP_Data_Type***
- #define: ***CRYP_DATATYPE_32B*** ((*uint32_t*)0x00000000)
 - #define: ***CRYP_DATATYPE_16B AES_CR_DATATYPE_0***
 - #define: ***CRYP_DATATYPE_8B AES_CR_DATATYPE_1***
 - #define: ***CRYP_DATATYPE_1B AES_CR_DATATYPE***

11 HAL DAC Generic Driver

11.1 DAC Firmware driver registers structures

11.1.1 DAC_HandleTypeDef

DAC_HandleTypeDef is defined in the `stm32l0xx_hal_dac.h`

Data Fields

- *DAC_TypeDef * Instance*
- *__IO HAL_DAC_StateTypeDef State*
- *HAL_LockTypeDef Lock*
- *DMA_HandleTypeDef * DMA_Handle1*
- *__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
- *__IO uint32_t DAC_HandleTypeDef::ErrorCode*
 - DAC Error code

11.1.2 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the `stm32l0xx_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](#)

11.1.3 DAC_TypeDef

DAC_TypeDef is defined in the stm32l052xx.h

Data Fields

- `__IO uint32_t CR`
- `__IO uint32_t SWTRIGR`
- `__IO uint32_t DHR12R1`
- `__IO uint32_t DHR12L1`
- `__IO uint32_t DHR8R1`
- `uint32_t RESERVED0`
- `__IO uint32_t DOR1`
- `uint32_t RESERVED1`
- `__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
- `uint32_t DAC_TypeDef::RESERVED0`
 - 0x14-0x28
- `__IO uint32_t DAC_TypeDef::DOR1`
 - DAC channel1 data output register, Address offset: 0x2C
- `uint32_t DAC_TypeDef::RESERVED1`
 - 0x30
- `__IO uint32_t DAC_TypeDef::SR`
 - DAC status register, Address offset: 0x34

11.2 DAC Firmware driver API description

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

11.2.1 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_DelInit\(\)*](#)
- [*HAL_DAC_MspInit\(\)*](#)
- [*HAL_DAC_MspDelInit\(\)*](#)

11.2.2 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\(\)*](#)

11.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.
- Set the specified data holding register value for Dual DAC channels.
- [*HAL_DAC_ConfigChannel\(\)*](#)
- [*HAL_DAC_SetValue\(\)*](#)

11.2.4 DAC Peripheral State functions

This subsection provides functions allowing to

- Check the DAC state.
- [*HAL_DAC_GetState\(\)*](#)
- [*HAL_DAC_IRQHandler\(\)*](#)
- [*HAL_DAC_GetError\(\)*](#)

11.2.4.1 HAL_DAC_Init

Function Name	HAL_StatusTypeDef HAL_DAC_Init (<i>DAC_HandleTypeDef</i> * hdac)
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	<ul style="list-style-type: none"> • hdac : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • HAL status

-
- | | |
|-------|---|
| Notes | <ul style="list-style-type: none"> None. |
|-------|---|

11.2.4.2 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	<ul style="list-style-type: none"> hdac : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

11.2.4.3 HAL_DAC_MspInit

Function Name	void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)
Function Description	Initializes the DAC MSP.
Parameters	<ul style="list-style-type: none"> hdac : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

11.2.4.4 HAL_DAC_MspDeInit

Function Name	void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)
Function Description	Deinitializes the DAC MSP.
Parameters	<ul style="list-style-type: none"> hdac : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

11.2.4.5 HAL_DAC_Start

Function Name	HAL_StatusTypeDef HAL_DAC_Start (<i>DAC_HandleTypeDef</i> *hdac, <i>uint32_t</i> channel)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> hdac : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC. channel : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 selected
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

11.2.4.6 HAL_DAC_Stop

Function Name	HAL_StatusTypeDef HAL_DAC_Stop (<i>DAC_HandleTypeDef</i> *hdac, <i>uint32_t</i> channel)
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> hdac : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC. channel : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 selected
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

11.2.4.7 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 using DMA.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 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: <ul style="list-style-type: none"> – 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.

11.2.4.8 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 using DMA.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 selected
Return values	• HAL status
Notes	• None.

11.2.4.9 HAL_DAC_GetValue

Function Name	<code>uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t channel)</code>
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 selected
Return values	• The selected DAC channel data output value.
Notes	<ul style="list-style-type: none"> • None.

11.2.4.10 HAL_DAC_ConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_DAC_ConfigChannel (</code> <code>DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef *</code> <code>sConfig, uint32_t channel)</code>
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 selected
Return values	• HAL status
Notes	<ul style="list-style-type: none"> • None.

11.2.4.11 HAL_DAC_SetValue

Function Name	<code>HAL_StatusTypeDef HAL_DAC_SetValue (</code> <code>DAC_HandleTypeDef * hdac, uint32_t channel, uint32_t</code> <code>alignment, uint32_t data)</code>
Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

	<ul style="list-style-type: none"> • alignment : Specifies the data alignment for DAC channel1. This parameter can be one of the following values: <ul style="list-style-type: none"> – 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.

11.2.4.12 HAL_DAC_GetState

Function Name	HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDefDef * hdac)
Function Description	return the DAC state
Parameters	<ul style="list-style-type: none"> • hdac : pointer to a DAC_HandleTypeDefDef structure that contains the configuration information for the specified DAC.
Return values	• HAL state
Notes	• None.

11.2.4.13 HAL_DAC_IRQHandler

Function Name	void HAL_DAC_IRQHandler (DAC_HandleTypeDefDef * hdac)
Function Description	Handles DAC interrupt request.
Parameters	<ul style="list-style-type: none"> • hdac : pointer to a DAC_HandleTypeDefDef structure that contains the configuration information for the specified DAC.
Return values	• None.
Notes	• None.

11.2.4.14 HAL_DAC_GetError

Function Name	<code>uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)</code>
Function Description	Return the DAC error code.
Parameters	<ul style="list-style-type: none"> • hdac : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • DAC Error Code
Notes	<ul style="list-style-type: none"> • None.

11.3 DAC Firmware driver defines

11.3.1 DAC

DAC

DAC_Channel_selection

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

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_flags_definition

- #define: `DAC_FLAG_DMAUDR1 ((uint32_t)DAC_SR_DMAUDR1)`
- #define: `DAC_IT_DMAUDR1 ((uint32_t)DAC_CR_DMAUDRIE1)`

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_T6_TRGO** ((*uint32_t*)DAC_CR_TEN1)

TIM6 TRGO selected as external conversion trigger for DAC channel

- #define: **DAC_TRIGGER_T21_TRGO** ((*uint32_t*)(DAC_CR_TSEL1_1 | DAC_CR_TSEL1_0 | DAC_CR_TEN1))

TIM21 TRGO selected as external conversion trigger for DAC channel

- #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_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

12 HAL DAC Extension Driver

12.1 DACEx Firmware driver introduction

12.2 DACEx Firmware driver API description

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

12.2.1 Peripheral Control functions

This section provides functions allowing to:

- Configure Triangle wave generation.
- Configure Noise wave generation.
- [***HAL_DACEx_TriangleWaveGenerate\(\)***](#)
- [***HAL_DACEx_NoiseWaveGenerate\(\)***](#)

12.2.1.1 [***HAL_DACEx_TriangleWaveGenerate***](#)

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (</code> <i>DAC_HandleTypeDef * hdac, uint32_t channel, uint32_t Amplitude</i>)
Function Description	Enables or disables the selected DAC channel wave triangle generation.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – <i>DAC_CHANNEL_1</i> : DAC Channel1 selected • Amplitude : Select max triangle amplitude. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>DAC_TRIANGLEAMPLITUDE_1</i> : Select max triangle amplitude of 1 – <i>DAC_TRIANGLEAMPLITUDE_3</i> : Select max triangle amplitude of 3 – <i>DAC_TRIANGLEAMPLITUDE_7</i> : Select max triangle amplitude of 7 – <i>DAC_TRIANGLEAMPLITUDE_15</i> : Select max triangle amplitude of 15 – <i>DAC_TRIANGLEAMPLITUDE_31</i> : Select max triangle amplitude of 31 – <i>DAC_TRIANGLEAMPLITUDE_63</i> : Select max triangle amplitude of 63 – <i>DAC_TRIANGLEAMPLITUDE_127</i> : Select max triangle amplitude of 127 – <i>DAC_TRIANGLEAMPLITUDE_255</i> : Select max triangle amplitude of 255

	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.

12.2.1.2 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 noise generation.
Parameters	<ul style="list-style-type: none"> • channel : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> – DAC_CHANNEL_1 : DAC Channel1 selected • Amplitude : Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: <ul style="list-style-type: none"> – 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

Return values	<ul style="list-style-type: none"> - 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
Notes	<ul style="list-style-type: none"> • HAL status • None.

12.3 DACEx Firmware driver defines

12.3.1 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

DACE_x_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)

13 HAL DMA Generic Driver

13.1 DMA Firmware driver registers structures

13.1.1 DMA_HandleTypeDef

DMA_HandleTypeDef is defined in the `stm32l0xx_hal_dma.h`

Data Fields

- *DMA_Channel_TypeDef * Instance*
- *DMA_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_DMA_StateTypeDef State*
- *void * Parent*
- *void(* XferCpltCallback*
- *void(* XferHalfCpltCallback*
- *void(* XferErrorCallback*
- *__IO uint32_t ErrorCode*

Field Documentation

- ***DMA_Channel_TypeDef* DMA_HandleTypeDef::Instance***
 - Register base address
- ***DMA_InitTypeDef DMA_HandleTypeDef::Init***
 - DMA communication parameters
- ***HAL_LockTypeDef DMA_HandleTypeDef::Lock***
 - DMA locking object
- ***__IO HAL_DMA_StateTypeDef DMA_HandleTypeDef::State***
 - DMA transfer state
- ***void* DMA_HandleTypeDef::Parent***
 - Parent object state
- ***void(* DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
 - DMA transfer complete callback
- ***void(* DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
 - DMA Half transfer complete callback
- ***void(* DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
 - DMA transfer error callback
- ***__IO uint32_t DMA_HandleTypeDef::ErrorCode***
 - DMA Error code

13.1.2 DMA_InitTypeDef

DMA_InitTypeDef is defined in the `stm32l0xx_hal_dma.h`

Data Fields

- *uint32_t Request*
- *uint32_t Direction*
- *uint32_t PeriphInc*
- *uint32_t MemInc*
- *uint32_t PeriphDataAlignment*
- *uint32_t MemDataAlignment*
- *uint32_t Mode*
- *uint32_t Priority*

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
 - Specifies the request selected for the specified channel. This parameter can be a value of [DMA_request](#)
- ***uint32_t DMA_InitTypeDef::Direction***
 - Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of [DMA_Data_transfer_direction](#)
- ***uint32_t DMA_InitTypeDef::PeriphInc***
 - Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of [DMA_Peripheral_incremented_mode](#)
- ***uint32_t DMA_InitTypeDef::MemInc***
 - Specifies whether the memory address register should be incremented or not. This parameter can be a value of [DMA_Memory_incremented_mode](#)
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
 - Specifies the Peripheral data width. This parameter can be a value of [DMA_Peripheral_data_size](#)
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
 - Specifies the Memory data width. This parameter can be a value of [DMA_Memory_data_size](#)
- ***uint32_t DMA_InitTypeDef::Mode***
 - Specifies the operation mode of the DMAy Channelx. This parameter can be a value of [DMA_mode](#)
- ***uint32_t DMA_InitTypeDef::Priority***
 - Specifies the software priority for the DMAy Channelx. This parameter can be a value of [DMA_Priority_level](#)

13.1.3 DMA_Channel_TypeDef

DMA_Channel_TypeDef is defined in the stm32l051xx.h

Data Fields

- *__IO uint32_t CCR*
- *__IO uint32_t CNDTR*
- *__IO uint32_t CPAR*
- *__IO uint32_t CMAR*

Field Documentation

- `__IO uint32_t DMA_Channel_TypeDef::CCR`
 - DMA channel x configuration register
- `__IO uint32_t DMA_Channel_TypeDef::CNDTR`
 - DMA channel x number of data register
- `__IO uint32_t DMA_Channel_TypeDef::CPAR`
 - DMA channel x peripheral address register
- `__IO uint32_t DMA_Channel_TypeDef::CMAR`
 - DMA channel x memory address register

13.1.4 DMA_TypeDef

`DMA_TypeDef` is defined in the `stm32l051xx.h`

Data Fields

- `__IO uint32_t ISR`
- `__IO uint32_t IFCR`

Field Documentation

- `__IO uint32_t DMA_TypeDef::ISR`
 - DMA interrupt status register, Address offset: 0x00
- `__IO uint32_t DMA_TypeDef::IFCR`
 - DMA interrupt flag clear register, Address offset: 0x04

13.1.5 DMA_Request_TypeDef

`DMA_Request_TypeDef` is defined in the `stm32l051xx.h`

Data Fields

- `__IO uint32_t CSELR`

Field Documentation

- `__IO uint32_t DMA_Request_TypeDef::CSELR`
 - DMA channel selection register, Address offset: 0xA8

13.2 DMA Firmware driver API description

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

13.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DMA
- De-Initialize the DMA
- [***HAL_DMA_Init\(\)***](#)
- [***HAL_DMA_DeInit\(\)***](#)

13.2.2 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\(\)***](#)

13.2.3 Peripheral State functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code
- [***HAL_DMA_GetState\(\)***](#)
- [***HAL_DMA_GetError\(\)***](#)

13.2.3.1 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	<ul style="list-style-type: none"> • hdma : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

13.2.3.2 HAL_DMA_DeInit

Function Name	HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)
Function Description	DeInitializes the DMA peripheral.
Parameters	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

13.2.3.3 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	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • SrcAddress : The source memory Buffer address • DstAddress : The destination memory Buffer address • DataLength : The length of data to be transferred from source to destination
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

13.2.3.4 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	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • SrcAddress : The source memory Buffer address • DstAddress : The destination memory Buffer address • DataLength : The length of data to be transferred from source to destination
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

13.2.3.5 HAL_DMA_Abort

Function Name	HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)
Function Description	Aborts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

13.2.3.6 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	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • CompleteLevel : Specifies the DMA level complete. • Timeout : Timeout duration.
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

13.2.3.7 HAL_DMA_IRQHandler

Function Name	void HAL_DMA_IRQHandler (<i>DMA_HandleTypeDef</i> * hdma)
Function Description	Handles DMA interrupt request.
Parameters	<ul style="list-style-type: none"> hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

13.2.3.8 HAL_DMA_GetState

Function Name	HAL_DMA_StateTypeDef HAL_DMA_GetState (<i>DMA_HandleTypeDef</i> * hdma)
Function Description	Returns the DMA state.
Parameters	<ul style="list-style-type: none"> hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> HAL state
Notes	<ul style="list-style-type: none"> None.

13.2.3.9 HAL_DMA_GetError

Function Name	uint32_t HAL_DMA_GetError (<i>DMA_HandleTypeDef</i> * hdma)
Function Description	Return the DMA error code.
Parameters	<ul style="list-style-type: none"> hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA

	Channel.
Return values	<ul style="list-style-type: none"> • DMA Error Code
Notes	<ul style="list-style-type: none"> • None.

13.3 DMA Firmware driver defines

13.3.1 DMA

DMA

DMA_Data_transfer_direction

- #define: **DMA_PERIPH_TO_MEMORY** ((uint32_t)0x00000000)
Peripheral to memory direction
- #define: **DMA_MEMORY_TO_PERIPH** ((uint32_t)DMA_CCR_DIR)
Memory to peripheral direction
- #define: **DMA_MEMORY_TO_MEMORY** ((uint32_t)(DMA_CCR_MEM2MEM))
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_TIMEOUT** ((uint32_t)0x00000020)
Timeout error

DMA_flag_definitions

- #define: **DMA_FLAG_GL1** ((uint32_t)0x00000001)
- #define: **DMA_FLAG_TC1** ((uint32_t)0x00000002)

- #define: **DMA_FLAG_HT1** ((*uint32_t*)0x00000004)
- #define: **DMA_FLAG_TE1** ((*uint32_t*)0x00000008)
- #define: **DMA_FLAG_GL2** ((*uint32_t*)0x00000010)
- #define: **DMA_FLAG_TC2** ((*uint32_t*)0x00000020)
- #define: **DMA_FLAG_HT2** ((*uint32_t*)0x00000040)
- #define: **DMA_FLAG_TE2** ((*uint32_t*)0x00000080)
- #define: **DMA_FLAG_GL3** ((*uint32_t*)0x00000100)
- #define: **DMA_FLAG_TC3** ((*uint32_t*)0x00000200)
- #define: **DMA_FLAG_HT3** ((*uint32_t*)0x00000400)
- #define: **DMA_FLAG_TE3** ((*uint32_t*)0x00000800)
- #define: **DMA_FLAG_GL4** ((*uint32_t*)0x00001000)
- #define: **DMA_FLAG_TC4** ((*uint32_t*)0x00002000)

- #define: **DMA_FLAG_HT4** ((*uint32_t*)0x00004000)
- #define: **DMA_FLAG_TE4** ((*uint32_t*)0x00008000)
- #define: **DMA_FLAG_GL5** ((*uint32_t*)0x00010000)
- #define: **DMA_FLAG_TC5** ((*uint32_t*)0x00020000)
- #define: **DMA_FLAG_HT5** ((*uint32_t*)0x00040000)
- #define: **DMA_FLAG_TE5** ((*uint32_t*)0x00080000)
- #define: **DMA_FLAG_GL6** ((*uint32_t*)0x00100000)
- #define: **DMA_FLAG_TC6** ((*uint32_t*)0x00200000)
- #define: **DMA_FLAG_HT6** ((*uint32_t*)0x00400000)
- #define: **DMA_FLAG_TE6** ((*uint32_t*)0x00800000)
- #define: **DMA_FLAG_GL7** ((*uint32_t*)0x01000000)
- #define: **DMA_FLAG_TC7** ((*uint32_t*)0x02000000)

- #define: **DMA_FLAG_HT7** ((*uint32_t*)0x04000000)
- #define: **DMA_FLAG_TE7** ((*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_TRIGGER** ((*uint16_t*) 0x5)

Index of the DMA handle used for Trigger DMA requests

DMA_interrupt_enable_definitions

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

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

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

DMA_Memory_data_size

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

Memory data alignment : Byte

- #define: **DMA_MDATAALIGN_HALFWORDx** ((*uint32_t*)DMA_CCR_MSIZEx_0)

Memory data alignment : HalfWord

- #define: **DMA_MDATAALIGN_WORDx** ((*uint32_t*)DMA_CCR_MSIZEx_1)

Memory data alignment : Word

DMA_Memory_incremented_mode

- #define: **DMA_MINC_ENABLEx** ((*uint32_t*)DMA_CCR_MINCx)

Memory increment mode Enable

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

Memory increment mode Disable

DMA_mode

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

Normal Mode

- #define: **DMA_CIRCULARx** ((*uint32_t*)DMA_CCR_CIRC)

Circular Mode

DMA_Peripheral_data_size

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

Peripheral data alignment : Byte

- #define: **DMA_PDATAALIGN_HALFWORDx** ((*uint32_t*)DMA_CCR_PSIZEx_0)

Peripheral data alignment : HalfWord

- #define: **DMA_PDATAALIGN_WORDx** ((*uint32_t*)DMA_CCR_PSIZEx_1)

Peripheral data alignment : Word

DMA_Peripheral_incremented_mode

- #define: **DMA_PINC_ENABLEx** ((*uint32_t*)DMA_CCR_PINCx)

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_CCR_PL_0)

Priority level : Medium

- #define: **DMA_PRIORITY_HIGH** ((*uint32_t*)DMA_CCR_PL_1)

Priority level : High

- #define: **DMA_PRIORITY VERY HIGH** ((*uint32_t*)DMA_CCR_PL)

Priority level : Very_High

DMA_request

- #define: **DMA_REQUEST_0** ((*uint32_t*)0x00000000)

- #define: **DMA_REQUEST_1** ((*uint32_t*)0x00000001)

- #define: **DMA_REQUEST_2** ((*uint32_t*)0x00000002)

- #define: **DMA_REQUEST_3** ((*uint32_t*)0x00000003)

- #define: **DMA_REQUEST_4** ((*uint32_t*)0x00000004)

- #define: **DMA_REQUEST_5** ((*uint32_t*)0x00000005)

- #define: **DMA_REQUEST_6** ((*uint32_t*)0x00000006)
- #define: **DMA_REQUEST_7** ((*uint32_t*)0x00000007)
- #define: **DMA_REQUEST_8** ((*uint32_t*)0x00000008)
- #define: **DMA_REQUEST_9** ((*uint32_t*)0x00000009)
- #define: **DMA_REQUEST_11** ((*uint32_t*)0x0000000B)

14 HAL FLASH Generic Driver

14.1 FLASH Firmware driver introduction

14.2 FLASH Firmware driver registers structures

14.2.1 **FLASH_EraseInitTypeDef**

FLASH_EraseInitTypeDef is defined in the `stm32l0xx_hal_flash.h`

Data Fields

- *uint32_t TypeErase*
- *uint32_t Page*
- *uint32_t NbPages*

Field Documentation

- *uint32_t FLASH_EraseInitTypeDef::TypeErase*
 - TypeErase: Mass erase or sector Erase. This parameter can be a value of `FLASH_Type_Erase`
- *uint32_t FLASH_EraseInitTypeDef::Page*
 - Sector: Initial FLASH sector to erase when Mass erase is disabled. This parameter must be an address value between 0x08000000 and 0x0800FFFF
- *uint32_t FLASH_EraseInitTypeDef::NbPages*
 - NbSectors: Number of sectors to be erased. This parameter must be a value between 1 and (max number of sectors - initial sector value)

14.2.2 **FLASH_OBProgramInitTypeDef**

FLASH_OBProgramInitTypeDef is defined in the `stm32l0xx_hal_flash.h`

Data Fields

- *uint32_t OptionType*
- *uint32_t WRPState*
- *uint32_t WRPSector*
- *uint32_t RDPLevel*
- *uint32_t BORLevel*
- *uint8_t USERConfig*

Field Documentation

- *uint32_t FLASH_OBProgramInitTypeDef::OptionType*

- OptionType: Option byte to be configured. This parameter can be a value of ***FLASH_Option_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState***
 - WRPState: Write protection activation or deactivation. This parameter can be a value of ***FLASH_WRP_State***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector***
 - WRPSector: specifies the sector(s) to be write protected. The value of this parameter depends on device used within the same series
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPLevel***
 - RDPLevel: Set the read protection level.. This parameter can be a value of ***FLASH_Option_Bytes_Read_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::BORLevel***
 - BORLevel: Set the BOR Level. This parameter can be a value of ***FLASH_Option_Bytes_BOR_Level***
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig***
 - USERConfig: Program the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY. This parameter can be a combination of ***FLASH_Option_Bytes_IWWatchdog***, ***FLASH_Option_Bytes_nRST_STOP*** and ***FLASH_Option_Bytes_nRST_STDBY***

14.2.3 **FLASH_ProcessTypeDef**

FLASH_ProcessTypeDef is defined in the `stm32l0xx_hal_flash.h`

Data Fields

- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t NbPagesToErase***
- ***__IO uint32_t Page***
- ***__IO uint32_t Address***
- ***HAL_LockTypeDef Lock***
- ***__IO FLASH_ErrorTypeDef ErrorCode***

Field Documentation

- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
- ***__IO uint32_t FLASH_ProcessTypeDef::NbPagesToErase***
- ***__IO uint32_t FLASH_ProcessTypeDef::Page***
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
- ***__IO FLASH_ErrorTypeDef FLASH_ProcessTypeDef::ErrorCode***

14.2.4 **FLASH_TypeDef**

FLASH_TypeDef is defined in the `stm32l051xx.h`

Data Fields

- ***__IO uint32_t ACR***

- `__IO uint32_t PECR`
- `__IO uint32_t PDKEYR`
- `__IO uint32_t PEKEYR`
- `__IO uint32_t PRGKEYR`
- `__IO uint32_t OPTKEYR`
- `__IO uint32_t SR`
- `__IO uint32_t OBR`
- `__IO uint32_t WRPR`

Field Documentation

- `__IO uint32_t FLASH_TypeDef::ACR`
 - Access control register, Address offset: 0x00
- `__IO uint32_t FLASH_TypeDef::PECR`
 - Program/erase control register, Address offset: 0x04
- `__IO uint32_t FLASH_TypeDef::PDKEYR`
 - Power down key register, Address offset: 0x08
- `__IO uint32_t FLASH_TypeDef::PEKEYR`
 - Program/erase key register, Address offset: 0x0c
- `__IO uint32_t FLASH_TypeDef::PRGKEYR`
 - Program memory key register, Address offset: 0x10
- `__IO uint32_t FLASH_TypeDef::OPTKEYR`
 - Option byte key register, Address offset: 0x14
- `__IO uint32_t FLASH_TypeDef::SR`
 - Status register, Address offset: 0x18
- `__IO uint32_t FLASH_TypeDef::OBR`
 - Option byte register, Address offset: 0x1c
- `__IO uint32_t FLASH_TypeDef::WRPR`
 - Write protection register, Address offset: 0x20

14.3 FLASH Firmware driver API description

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

14.3.1 FLASH peripheral features

The Flash memory interface manages CPU accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option Bytes programming

14.3.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32L0xx 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 and 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_GetErrorCode()

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 preread buffer
 - Enable/Disable the Flash power-down
 - Enable/Disable the FLASH interrupts
 - Monitor the FLASH flags status
-
-

14.3.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

The FLASH Memory Programming functions, includes the following functions:

- HAL_FLASH_Unlock(void);
- HAL_FLASH_Lock(void);
- HAL_FLASH_Program(uint32_t TypeProgram, uint32_t Address, uint32_t Data)
- HAL_FLASH_Program_IT(uint32_t TypeProgram, uint32_t Address, uint32_t Data)

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_Unlock() function to enable the flash control register and program memory access.
 2. Call the desired function to erase page or program data.
 3. Call the HAL_FLASH_Lock() to disable the flash program memory access (recommended to protect the FLASH memory against possible unwanted operation).
-
-

14.3.4 Option Bytes Programming functions

The FLASH_Option Bytes Programming_functions, includes the following functions:

- HAL_FLASH_OB_Unlock(void);
- HAL_FLASH_OB_Lock(void);
- HAL_FLASH_OB_Launch(void);

- HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
- HAL_FLASHEx_OBGetConfig(FLASH_OBProgramInitTypeDef *pOBInit);

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_OB_Unlock() function to enable the Flash option control register access.
2. Call the following functions to program the desired option bytes.
 - HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
3. Once all needed option bytes to be programmed are correctly written, call the HAL_FLASH_OB_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection (nWRPi bits). As a result, these 2 options are exclusive each other.
2. In order to activate the PcROP (change the function of the nWRPi option bits), the SPRMOD option bit must be activated.
3. The active value of nWRPi bits is inverted when PCROP mode is active, this means: if SPRMOD = 1 and nWRPi = 1 (default value), then the user page "i" is read/write protected.
4. To activate PCROP mode for Flash page(s), you need to follow the sequence below:
 - For page(s) within the first 64KB of the Flash, use this function HAL_FLASHEx_AdvOBProgram with PCROPState = PCROPSTATE_ENABLE. *

14.3.5 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\(\)**](#)

14.3.6 Peripheral Errors functions

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

- [**HAL_FLASH_GetError\(\)**](#)

14.3.6.1 HAL_FLASH_Program

Function Name	HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)
Function Description	FLASH memory functions that can be executed from FLASH.
Parameters	<ul style="list-style-type: none"> • TypeProgram : Indicate the way to program at a specified address. This parameter can be a value of

	FLASH_Type_Program
Return values	<ul style="list-style-type: none"> • Address : specifies the address to be programmed. • Data : specifies the data to be programmed
Notes	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
	None.

14.3.6.2 HAL_FLASH_Program_IT

Function Name	HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint32_t Data)
Function Description	Program word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
Notes	None.

14.3.6.3 HAL_FLASH_IRQHandler

Function Name	void HAL_FLASH_IRQHandler (void)
Function Description	This function handles FLASH interrupt request.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.

Notes

None.

14.3.6.4 HAL_FLASH_EndOfOperationCallback

Function Name	void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)
Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue : The value saved in this parameter depends on the ongoing procedure Pages_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	<ul style="list-style-type: none"> • none
Notes	<ul style="list-style-type: none"> • None.

14.3.6.5 HAL_FLASH_OperationErrorCallback

Function Name	void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)
Function Description	FLASH operation error interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue : The value saved in this parameter depends on the ongoing procedure Pages_Erase: Sector number which returned an error Program: Address which was selected for data program
Return values	<ul style="list-style-type: none"> • none
Notes	<ul style="list-style-type: none"> • None.

14.3.6.6 HAL_FLASH_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASH_Unlock (void)
Function Description	Unlock the FLASH control register access.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none"> • None.

14.3.6.7 HAL_FLASH_Lock

Function Name	HAL_StatusTypeDef HAL_FLASH_Lock (void)
Function Description	Locks the FLASH control register access.
Parameters	<ul style="list-style-type: none">None.
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none">None.

14.3.6.8 HAL_FLASH_OB_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)
Function Description	Unlock the FLASH Option Control Registers access.
Parameters	<ul style="list-style-type: none">None.
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none">None.

14.3.6.9 HAL_FLASH_OB_Lock

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)
Function Description	Lock the FLASH Option Control Registers access.
Parameters	<ul style="list-style-type: none">None.
Return values	<ul style="list-style-type: none">HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none">None.

14.3.6.10 HAL_FLASH_OB_Launch

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)
Function Description	Launch the option byte loading.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none"> None.

14.3.6.11 HAL_FLASH_GetError

Function Name	FLASH_ErrorTypeDef HAL_FLASH_GetError (void)
Function Description	Get the specific FLASH error flag.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> FLASH_ErrorCode : The returned value can be: <ul style="list-style-type: none"> FLASH_ERROR_RD: <i>FLASH Read Protection error flag (PCROP)</i> FLASH_ERROR_ENDHV: <i>FLASH Programming Sequence error flag</i> FLASH_ERROR_SIZE: <i>FLASH Programming Parallelism error flag</i> FLASH_ERROR_PGA: <i>FLASH Programming Alignment error flag</i> FLASH_ERROR_WRP: <i>FLASH Write protected error flag</i> FLASH_ERROR_OPTV: <i>FLASH Option valid error flag</i> FLASH_ERROR_NOTZERO: <i>FLASH write operation is done in a not-erased region</i>
Notes	<ul style="list-style-type: none"> None.

14.4 FLASH Firmware driver defines

14.4.1 FLASH

FLASH

FLASH_Flags

- #define: **FLASH_FLAG_BSY** **FLASH_SR_BSY**
FLASH Busy flag



- #define: **FLASH_FLAG_EOP** **FLASH_SR_EOP**
FLASH End of Programming flag
- #define: **FLASH_FLAG_ENDHV** **FLASH_SR_ENHV**
FLASH End of High Voltage flag
- #define: **FLASH_FLAG_READY** **FLASH_SR_READY**
FLASH Ready flag after low power mode
- #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_SIZERR** **FLASH_SR_SIZERR**
FLASH Size error flag
- #define: **FLASH_FLAG_OPTVERR** **FLASH_SR_OPTVERR**
FLASH Option Validity error flag
- #define: **FLASH_FLAG_RDERR** **FLASH_SR_RDERR**
FLASH Read protected error flag
- #define: **FLASH_FLAG_NOTZEROERR** **FLASH_SR_NOTZEROERR**
FLASH Read protected error flag

FLASH_Interrupts

- #define: **FLASH_IT_EOP** **FLASH_PECR_EOPIE**
End of programming interrupt source
- #define: **FLASH_IT_ERR** **FLASH_PECR_ERRIE**
Error interrupt source

FLASH_Keys

- #define: **FLASH_PDKEY1** ((uint32_t)0x04152637)

Flash power down key1

- #define: **FLASH_PDKEY2** ((uint32_t)0xAFBFCFD)

Flash power down key2: used with FLASH_PDKEY1 to unlock the RUN_PD bit in FLASH_ACR

- #define: **FLASH_PEKEY1** ((uint32_t)0x89ABCDEF)

Flash program erase key1

- #define: **FLASH_PEKEY2** ((uint32_t)0x02030405)

Flash program erase key: used with FLASH_PEKEY2 to unlock the write access to the FLASH_PECR register and data EEPROM

- #define: **FLASH_PRGKEY1** ((uint32_t)0x8C9DAEBF)

Flash program memory key1

- #define: **FLASH_PRGKEY2** ((uint32_t)0x13141516)

Flash program memory key2: used with FLASH_PRGKEY2 to unlock the program memory

- #define: **FLASH_OPTKEY1** ((uint32_t)0xFBead9C8)

Flash option key1

- #define: **FLASH_OPTKEY2** ((uint32_t)0x24252627)

Flash option key2: used with FLASH_OPTKEY1 to unlock the write access to the option byte block

FLASH_Option_Bytes_BOR_Level

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

BOR is disabled at power down, the reset is asserted when the VDD power supply reaches the PDR(Power Down Reset) threshold (1.5V)

- #define: **OB_BOR_LEVEL1** ((uint8_t)0x08)

BOR Reset threshold levels for 1.7V - 1.8V VDD power supply

- #define: **OB_BOR_LEVEL2** ((uint8_t)0x09)

BOR Reset threshold levels for 1.9V - 2.0V VDD power supply

- #define: ***OB_BOR_LEVEL3 ((uint8_t)0x0A)***
BOR Reset threshold levels for 2.3V - 2.4V VDD power supply

- #define: ***OB_BOR_LEVEL4 ((uint8_t)0x0B)***
BOR Reset threshold levels for 2.55V - 2.65V VDD power supply

- #define: ***OB_BOR_LEVEL5 ((uint8_t)0x0C)***
BOR Reset threshold levels for 2.8V - 2.9V VDD power supply

FLASH_Option_Bytes_IWatchdog

- #define: ***OB_IWDG_SW ((uint8_t)0x10)***
Software WDG selected

- #define: ***OB_IWDG_HW ((uint8_t)0x00)***
Hardware WDG selected

FLASH_Option_Bytes_nRST_STDBY

- #define: ***OB_STDBY_NoRST ((uint8_t)0x40)***
No reset generated when entering in STANDBY

- #define: ***OB_STDBY_RST ((uint8_t)0x00)***
Reset generated when entering in STANDBY

FLASH_Option_Bytes_nRST_STOP

- #define: ***OB_STOP_NoRST ((uint8_t)0x20)***
No reset generated when entering in STOP

- #define: ***OB_STOP_RST ((uint8_t)0x00)***
Reset generated when entering in STOP

FLASH_Option_Bytes_Read_Protection

- #define: ***OB_RDP_Level_0 ((uint8_t)0xAA)***

- #define: ***OB_RDP_Level_1 ((uint8_t)0xBB)***

FLASH_Option_Bytes_Write_Protection

- #define: *OB_WRP_Pages0to31* ((*uint32_t*)0x00000001)
- #define: *OB_WRP_Pages32to63* ((*uint32_t*)0x00000002)
- #define: *OB_WRP_Pages64to95* ((*uint32_t*)0x00000004)
- #define: *OB_WRP_Pages96to127* ((*uint32_t*)0x00000008)
- #define: *OB_WRP_Pages128to159* ((*uint32_t*)0x00000010)
- #define: *OB_WRP_Pages160to191* ((*uint32_t*)0x00000020)
- #define: *OB_WRP_Pages192to223* ((*uint32_t*)0x00000040)
- #define: *OB_WRP_Pages224to255* ((*uint32_t*)0x00000080)
- #define: *OB_WRP_Pages256to287* ((*uint32_t*)0x00000100)
- #define: *OB_WRP_Pages288to319* ((*uint32_t*)0x00000200)
- #define: *OB_WRP_Pages320to351* ((*uint32_t*)0x00000400)
- #define: *OB_WRP_Pages352to383* ((*uint32_t*)0x00000800)

- #define: ***OB_WRP_Pages384to415*** ((*uint32_t*)0x00001000)
- #define: ***OB_WRP_Pages416to447*** ((*uint32_t*)0x00002000)
- #define: ***OB_WRP_Pages448to479*** ((*uint32_t*)0x00004000)
- #define: ***OB_WRP_Pages480to511*** ((*uint32_t*)0x00008000)
- #define: ***OB_WRP_AllPages*** ((*uint32_t*)0x0000FFFF)
Write protection of all Sectors

FLASH_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

FLASH_Type_Erase

- #define: ***TYPEERASE_PAGEERASE*** ((*uint32_t*)0x00)
Page erase only
- #define: ***TYPEERASE_WORD*** ((*uint32_t*)0x01)
Data erase word activation

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_FASTBYTE ((uint32_t)0x04)***
Fast Program byte (8-bit) at a specified address.

- #define: ***TYPEPROGRAM_FASTHALFWORD ((uint32_t)0x08)***
Fast Program a half-word (16-bit) at a specified address.

- #define: ***TYPEPROGRAM_FASTWORD ((uint32_t)0x10)***
Fast Program a word (32-bit) at a specified address.

FLASH_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

15 HAL FLASH Extension Driver

15.1 FLASHEx Firmware driver introduction

15.2 FLASHEx Firmware driver registers structures

15.2.1 `FLASH_AdvOBProgramInitTypeDef`

`FLASH_AdvOBProgramInitTypeDef` is defined in the `stm32l0xx_hal_flash_ex.h`

Data Fields

- `uint32_t OptionType`
- `uint32_t PCROPState`
- `uint16_t Pages`
- `uint16_t BootConfig`

Field Documentation

- `uint32_t FLASH_AdvOBProgramInitTypeDef::OptionType`
 - OptionType: Option byte to be configured for extension . This parameter can be a value of `FLASHEx_Option_Type`
- `uint32_t FLASH_AdvOBProgramInitTypeDef::PCROPState`
 - PCROPState: PCROP activation or deactivation. This parameter can be a value of `FLASHEx_PCROP_State`
- `uint16_t FLASH_AdvOBProgramInitTypeDef::Pages`
 - Sectors: specifies the sector(s) set for PCROP This parameter can be a value of `FLASHEx_Option_Bytes_PC_ReadWrite_Protection`
- `uint16_t FLASH_AdvOBProgramInitTypeDef::BootConfig`
 - BootConfig: specifies Option bytes for boot config This parameter can be a value of `FLASHEx_Option_Bytes_BOOT1`

15.3 FLASHEx Firmware driver API description

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

15.3.1 Flash peripheral Extended features

Comparing to other products, the FLASH interface for STM32L0xx devices contains the following additional features

- DATA EEPROM memory management
- BOOT option bit configuration
- PCROP protection for all sectors

15.3.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32L0xx. It includes:

- Full DATA_EEPROM erase and program management
- Boot activation
- PCROP protection configuration and control for all pages

15.3.3 Extended Features functions

This subsection provides a set of functions allowing to manage the FLASH data transfers.

- [*HAL_FLASHEx_AdvOBProgram\(\)*](#)
- [*HAL_FLASHEx_AdvOBGetConfig\(\)*](#)

15.3.4 DATA EEPROM Programming functions

The DATA_EEPROM Programming_Functions, includes the following functions:

- `HAL_DATA_EEPROMEx_Unlock(void);`
- `HAL_DATA_EEPROMEx_Lock(void);`
- `HAL_DATA_EEPROMEx_Erase(uint32_t Address)`
- `HAL_DATA_EEPROMEx_Program(uint32_t TypeProgram, uint32_t Address, uint32_t Data)`

Any operation of erase or program should follow these steps:

1. Call the `HAL_DATA_EEPROMEx_Unlock()` function to enable the data EEPROM access and Flash program erase control register access.
 2. Call the desired function to erase or program data.
 3. Call the `HAL_DATA_EEPROMEx_Lock()` to disable the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).
- [*HAL_DATA_EEPROMEx_Unlock\(\)*](#)
 - [*HAL_DATA_EEPROMEx_Lock\(\)*](#)
 - [*HAL_DATA_EEPROMEx_Erase\(\)*](#)
 - [*HAL_DATA_EEPROMEx_Program\(\)*](#)
 - [*HAL_FLASHEx_Erase\(\)*](#)
 - [*HAL_FLASHEx_Erase_IT\(\)*](#)
 - [*HAL_FLASHEx_OBProgram\(\)*](#)
 - [*HAL_FLASHEx_OBGetConfig\(\)*](#)

15.3.4.1 HAL_FLASHEx_AdvOBProgram

Function Name	<code>HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)</code>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> • <code>pAdvOBInit</code> : pointer to an <code>FLASH_AdvOBProgramInitTypeDef</code> structure that contains

the configuration information for the programming.

- | | |
|---------------|---|
| Return values | <ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status |
| Notes | <ul style="list-style-type: none"> • None. |

15.3.4.2 HAL_FLASHEx_AdvOBGetConfig

Function Name	void HAL_FLASHEx_AdvOBGetConfig (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)
Function Description	Get the OBEX byte configuration.
Parameters	<ul style="list-style-type: none"> • pAdvOBInit : pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

15.3.4.3 HAL_DATA_EEPROMEx_Unlock

Function Name	HAL_StatusTypeDef HAL_DATA EEPROMEx Unlock (void)
Function Description	Unlocks the data memory and FLASH_PECR register access.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status

- Notes
- None.

15.3.4.4 HAL_DATA_EEPROMEx_Lock

Function Name	HAL_StatusTypeDef HAL_DATA EEPROMEx Lock (void)
Function Description	Locks the Data memory and FLASH_PECR register access.

Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none"> • None.

15.3.4.5 HAL_DATA_EEPROMEx_Erase

Function Name	HAL_StatusTypeDef HAL_DATA_EEPROMEx_Erase (uint32_t Address)
Function Description	Erase a word in data memory.
Parameters	<ul style="list-style-type: none"> • Address : specifies the address to be erased.
Return values	<ul style="list-style-type: none"> • FLASH Status: The returned value can be: FLASH_ERROR_PROGRAM, FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
Notes	<ul style="list-style-type: none"> • To correctly run this function, the DATA EEPROM Unlock() function must be called before. Call the DATA EEPROM Lock() to the data EEPROM access and Flash program erase control register access(recommended to protect the DATA EEPROM against possible unwanted operation).

15.3.4.6 HAL_DATA_EEPROMEx_Program

Function Name	HAL_StatusTypeDef HAL_DATA_EEPROMEx_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)
Function Description	Program word at a specified address.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none"> • None.

15.3.4.7 HAL_FLASHEx_Erase

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase (<i>FLASH_EraselInitTypeDef</i> * pEraselInit, uint32_t * PageError)
Function Description	Erase the specified FLASH memory Pages.
Parameters	<ul style="list-style-type: none"> • pEraselInit : pointer to an <i>FLASH_EraselInitTypeDef</i> structure that contains the configuration information for the erasing. • PageError : 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	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • None.

15.3.4.8 HAL_FLASHEx_Erase_IT

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (<i>FLASH_EraselInitTypeDef</i> * pEraselInit)
Function Description	Perform a page erase of the specified FLASH memory pages with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • pEraselInit : pointer to an <i>FLASH_EraselInitTypeDef</i> structure that contains the configuration information for the erasing.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • None.

15.3.4.9 HAL_FLASHEx_OBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OBProgram (<i>FLASH_OBProgramInitTypeDef</i> * pOBInit)
---------------	---

Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> • pOBInit : pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none"> • None.

15.3.4.10 HAL_FLASHEx_OBGetConfig

Function Name	void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Get the Option byte configuration.
Parameters	<ul style="list-style-type: none"> • pOBInit : pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

15.4 FLASHEx Firmware driver defines

15.4.1 FLASHEx

FLASHEx

FLASHEx_Latency

- #define: **FLASH_LATENCY_0 ((uint8_t)0x00)**
FLASH Zero Latency cycle
- #define: **FLASH_LATENCY_1 FLASH_ACR_LATENCY**
FLASH One Latency cycle

FLASHEx_Macros

- #define: **__HAL_FLASH_PREFETCH_BUFFER_ENABLE (FLASH->ACR |= FLASH_ACR_PRFTEN)**
none

- #define: **`__HAL_FLASH_PREFETCH_BUFFER_DISABLE (FLASH->ACR &= (~FLASH_ACR_PRFTEN))`**
none

- #define: **`__HAL_FLASH_BUFFER_CACHE_ENABLE (FLASH->ACR &= (~FLASH_ACR_DISAB_BUF))`**
none

- #define: **`__HAL_FLASH_BUFFER_CACHE_DISABLE (FLASH->ACR |= FLASH_ACR_DISAB_BUF)`**
none

- #define: **`__HAL_FLASH_PREREAD_BUFFER_ENABLE (FLASH->ACR |= FLASH_ACR_PRE_READ)`**
none

- #define: **`__HAL_FLASH_PREREAD_BUFFER_DISABLE (FLASH->ACR &= (~FLASH_ACR_PRE_READ))`**
none

- #define: **`__HAL_FLASH_SLEEP_POWERDOWN_ENABLE SET_BIT(FLASH->ACR, FLASH_ACR_SLEEP_PD)`**
none

- #define: **`__HAL_FLASH_SLEEP_POWERDOWN_DISABLE CLEAR_BIT(FLASH->ACR, FLASH_ACR_SLEEP_PD)`**
none

- #define: **`__HAL_FLASH_POWER_DOWN_ENABLE do { FLASH->PDKEYR = FLASH_PDKEY1; \ FLASH->PDKEYR = FLASH_PDKEY2; \ SET_BIT(FLASH->ACR, FLASH_ACR_RUN_PD); \ } while (0)`**
Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.

- #define: **`__HAL_FLASH_POWER_DOWN_DISABLE do { FLASH->PDKEYR = FLASH_PDKEY1; \ FLASH->PDKEYR = FLASH_PDKEY2; \ CLEAR_BIT(FLASH->ACR, FLASH_ACR_RUN_PD); \ } while (0)`**

FLASHEx_Option_Bytes_BOOT1

-
- #define: ***OB_BOOT1_RESET*** ((*uint16_t*)0x0000)
BOOT1 Reset
 - #define: ***OB_BOOT1_SET*** ((*uint16_t*)0x8000)
BOOT1 Set
- FLASHEx_Option_Bytes_PC_ReadWrite_Protection*
- #define: ***OB_PCROP_Pages0to31*** ((*uint32_t*)0x00000001)
 - #define: ***OB_PCROP_Pages32to63*** ((*uint32_t*)0x00000002)
 - #define: ***OB_PCROP_Pages64to95*** ((*uint32_t*)0x00000004)
 - #define: ***OB_PCROP_Pages96to127*** ((*uint32_t*)0x00000008)
 - #define: ***OB_PCROP_Pages128to159*** ((*uint32_t*)0x00000010)
 - #define: ***OB_PCROP_Pages160to191*** ((*uint32_t*)0x00000020)
 - #define: ***OB_PCROP_Pages192to223*** ((*uint32_t*)0x00000040)
 - #define: ***OB_PCROP_Pages224to255*** ((*uint32_t*)0x00000080)
 - #define: ***OB_PCROP_Pages256to287*** ((*uint32_t*)0x00000100)
 - #define: ***OB_PCROP_Pages288to319*** ((*uint32_t*)0x00000200)

- #define: ***OB_PCROP_Pages320to351*** ((*uint32_t*)0x00000400)
- #define: ***OB_PCROP_Pages352to383*** ((*uint32_t*)0x00000800)
- #define: ***OB_PCROP_Pages384to415*** ((*uint32_t*)0x00001000)
- #define: ***OB_PCROP_Pages416to447*** ((*uint32_t*)0x00002000)
- #define: ***OB_PCROP_Pages448to479*** ((*uint32_t*)0x00004000)
- #define: ***OB_PCROP_Pages480to511*** ((*uint32_t*)0x00008000)
- #define: ***OB_PCROP_AllPages*** ((*uint32_t*)0x0000FFFF)
PC Read/Write protection of all Sectors

FLASHEx_Option_Type

- #define: ***OBEX_PCROP*** ((*uint32_t*)0x01)
PCROP option byte configuration
- #define: ***OBEX_BOOTCONFIG*** ((*uint32_t*)0x02)
BOOTConfig option byte configuration

FLASHEx_PCROP_State

- #define: ***PCROPSTATE_DISABLE*** ((*uint32_t*)0x00)
Disable PCROP
- #define: ***PCROPSTATE_ENABLE*** ((*uint32_t*)0x01)
Enable PCROP

FLASHEx_Selection_Protection_Mode

- #define: ***OB_PCROP_DESELECTED ((uint16_t)0x0000)***

Disabled PcROP, nWPRi bits used for Write Protection on sector i

- #define: ***OB_PCROP_SELECTED ((uint16_t)0x0100)***

Enable PcROP, nWPRi bits used for PCRoP Protection on sector i

16 HAL FLASH Ram Function Driver

16.1 FLASHRamfunc Firmware driver introduction

16.2 FLASHRamfunc Firmware driver API description

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

16.2.1 ramfunc functions

This subsection provides a set of functions that should be executed from RAM transfers.

- *[FLASH_HalfPageProgram\(\)](#)*
- *[FLASH_EnableRunPowerDown\(\)](#)*
- *[FLASH_DisableRunPowerDown\(\)](#)*

16.2.1.1 **FLASH_HalfPageProgram**

Function Name	<code>__RAM_FUNC FLASH_HalfPageProgram (uint32_t Address, uint32_t * Data)</code>
Function Description	Program a half page word at a specified address.,
Parameters	<ul style="list-style-type: none"> • Address : specifies the address to be programmed, the address should be half page aligned. • *Data : specifies the buffer of data to be programmed, the size of the buffer is 16 words.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL_Status
Notes	<ul style="list-style-type: none"> • This function should be executed from RAM

16.2.1.2 **FLASH_EnableRunPowerDown**

Function Name	<code>__RAM_FUNC FLASH_EnableRunPowerDown (void)</code>
Function Description	Enable the Power down in Run Mode.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • This function should be called and executed from SRAM memory

16.2.1.3 **FLASH_DisableRunPowerDown**

Function Name	<code>__RAM_FUNC FLASH_DisableRunPowerDown (void)</code>
Function Description	Disable the Power down in Run Mode.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• This function should be called and executed from SRAM memory

16.3 **FLASHRamfunc Firmware driver defines**

16.3.1 **FLASHRamfunc**

FLASHRamfunc

17 HAL GPIO Generic Driver

17.1 GPIO Firmware driver introduction

17.2 GPIO Firmware driver registers structures

17.2.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the `stm32l0xx_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 [*GPIOEx_Alternate_function_selection*](#)

17.2.2 GPIO_TypeDef

GPIO_TypeDef is defined in the `stm32l051xx.h`

Data Fields

- *__IO uint32_t MODER*
- *__IO uint16_t OTYPER*
- *uint16_t RESERVED0*
- *__IO uint32_t OSPEEDR*

- `__IO uint32_t PUPDR`
- `__IO uint16_t IDR`
- `uint16_t RESERVED1`
- `__IO uint16_t ODR`
- `uint16_t RESERVED2`
- `__IO uint32_t BSRR`
- `__IO uint32_t LCKR`
- `__IO uint32_t AFR`
- `__IO uint16_t BRR`

Field Documentation

- `__IO uint32_t GPIO_TypeDef::MODER`
 - GPIO port mode register, Address offset: 0x00
- `__IO uint16_t GPIO_TypeDef::OTYPER`
 - GPIO port output type register, Address offset: 0x04
- `uint16_t GPIO_TypeDef::RESERVED0`
 - Reserved, 0x06
- `__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 uint16_t GPIO_TypeDef::IDR`
 - GPIO port input data register, Address offset: 0x10
- `uint16_t GPIO_TypeDef::RESERVED1`
 - Reserved, 0x12
- `__IO uint16_t GPIO_TypeDef::ODR`
 - GPIO port output data register, Address offset: 0x14
- `uint16_t GPIO_TypeDef::RESERVED2`
 - Reserved, 0x16
- `__IO uint32_t GPIO_TypeDef::BSRR`
 - GPIO port bit set/reset registerBSRR, Address offset: 0x18
- `__IO uint32_t GPIO_TypeDef::LCKR`
 - GPIO port configuration lock register, Address offset: 0x1C
- `__IO uint32_t GPIO_TypeDef::AFR`
 - GPIO alternate function register, Address offset: 0x20-0x24
- `__IO uint16_t GPIO_TypeDef::BRR`
 - GPIO bit reset register, Address offset: 0x28

17.3 GPIO Firmware driver API description

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

17.3.1 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.

17.3.2 How to use this driver

1. Enable the GPIO IOPORT clock using the following function:
`_GPIOx_CLK_ENABLE()`.
2. In case of external interrupt/event mode selection, enable the SYSCFG clock using the following function `_SYSCFG_CLK_ENABLE()`.
3. 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).
4. 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()`.
5. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
6. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin() / HAL_GPIO_TogglePin()`.
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).

-
8. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
 9. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

17.3.3 Initialization and de-initialization functions

- `HAL_GPIO_Init()`
- `HAL_GPIO_DeInit()`

17.3.4 IO operation functions

- `HAL_GPIO_ReadPin()`
- `HAL_GPIO_WritePin()`
- `HAL_GPIO_TogglePin()`
- `HAL_GPIO_EXTI_IRQHandler()`
- `HAL_GPIO_EXTI_Callback()`

17.3.4.1 HAL_GPIO_Init

Function Name	<code>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
Parameters	<ul style="list-style-type: none"> • GPIOx : where x can be (A..D and H) to select the GPIO peripheral for STM32L0XX family devices. • GPIO_InitStruct : pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

17.3.4.2 HAL_GPIO_DeInit

Function Name	<code>void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</code>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.

Parameters	<ul style="list-style-type: none"> GPIOx : where x can be (A..D and H) to select the GPIO peripheral for STM32L0XX family 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) except for GPIOD(2) and GPIOH(1:0).
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

17.3.4.3 HAL_GPIO_ReadPin

Function Name	GPIO_PinState HAL_GPIO_ReadPin (<i>GPIO_TypeDef</i> * GPIOx, uint16_t GPIO_Pin)
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> GPIOx : where x can be (A..D and H) to select the GPIO peripheral for STM32L0xx family devices. GPIO_Pin : specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15) except for GPIOD(2) and GPIOH(1:0).
Return values	<ul style="list-style-type: none"> The input port pin value.
Notes	<ul style="list-style-type: none"> None.

17.3.4.4 HAL_GPIO_WritePin

Function Name	void HAL_GPIO_WritePin (<i>GPIO_TypeDef</i> * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
Function Description	Sets or clears the selected data port bit.
Parameters	<ul style="list-style-type: none"> GPIOx : where x can be (A..D and H) to select the GPIO peripheral for STM32L0xx family 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) except for GPIOD(2) and GPIOH(1:0). PinState : specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pin GPIO_PIN_SET: to set the port pin

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> 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.

17.3.4.5 HAL_GPIO_TogglePin

Function Name	void HAL_GPIO_TogglePin (<i>GPIO_TypeDef</i> * GPIOx, uint16_t GPIO_Pin)
Function Description	Toggles the specified GPIO pins.
Parameters	<ul style="list-style-type: none"> GPIOx : Where x can be (A..D and H) to select the GPIO peripheral for STM32L0xx family devices. GPIO_Pin : Specifies the pins to be toggled.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

17.3.4.6 HAL_GPIO_EXTI_IRQHandler

Function Name	void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)
Function Description	This function handles EXTI interrupt request.
Parameters	<ul style="list-style-type: none"> GPIO_Pin : Specifies the pins connected EXTI line
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

17.3.4.7 HAL_GPIO_EXTI_Callback

Function Name	void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
Function Description	EXTI line detection callbacks.

Parameters	<ul style="list-style-type: none"> • GPIO_Pin : Specifies the pins connected EXTI line
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

17.4 GPIO Firmware driver defines

17.4.1 GPIO

GPIO

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

18 HAL GPIO Extension Driver

18.1 GPIOEx Firmware driver defines

18.1.1 GPIOEx

GPIOEx

GPIOEx_Alternate_function_selection

- #define: **GPIO_AF0_SPI1** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_SPI2** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_USART1** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_USART2** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_LPUART1** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_USB** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_LPTIM** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_TSC** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_TIM2** ((*uint8_t*)0x00)
- #define: **GPIO_AF0_TIM21** ((*uint8_t*)0x00)

- #define: ***GPIO_AF0_TIM22*** ((*uint8_t*)0x00)
- #define: ***GPIO_AF0_EVENTOUT*** ((*uint8_t*)0x00)
- #define: ***GPIO_AF0_MCO*** ((*uint8_t*)0x00)
- #define: ***GPIO_AF0_SWDIO*** ((*uint8_t*)0x00)
- #define: ***GPIO_AF0_SWCLK*** ((*uint8_t*)0x00)
- #define: ***GPIO_AF1_SPI1*** ((*uint8_t*)0x01)
- #define: ***GPIO_AF1_SPI2*** ((*uint8_t*)0x01)
- #define: ***GPIO_AF1_I2C1*** ((*uint8_t*)0x01)
- #define: ***GPIO_AF1_LCD*** ((*uint8_t*)0x01)
- #define: ***GPIO_AF1_TIM2*** ((*uint8_t*)0x01)
- #define: ***GPIO_AF1_TIM21*** ((*uint8_t*)0x01)
- #define: ***GPIO_AF2_SPI2*** ((*uint8_t*)0x02)

- #define: **GPIO_AF2_LPUART1** ((*uint8_t*)0x02)
- #define: **GPIO_AF2_USB** ((*uint8_t*)0x02)
- #define: **GPIO_AF2_LPTIM** ((*uint8_t*)0x02)
- #define: **GPIO_AF2_TIM2** ((*uint8_t*)0x02)
- #define: **GPIO_AF2_TIM22** ((*uint8_t*)0x02)
- #define: **GPIO_AF2_EVENTOUT** ((*uint8_t*)0x02)
- #define: **GPIO_AF2_RTC_50Hz** ((*uint8_t*)0x02)
- #define: **GPIO_AF3_I2C1** ((*uint8_t*)0x03)
- #define: **GPIO_AF3_TSC** ((*uint8_t*)0x03)
- #define: **GPIO_AF3_EVENTOUT** ((*uint8_t*)0x03)
- #define: **GPIO_AF4_I2C1** ((*uint8_t*)0x04)
- #define: **GPIO_AF4_USART1** ((*uint8_t*)0x04)

- #define: **GPIO_AF4_USART2** ((*uint8_t*)0x04)
- #define: **GPIO_AF4_LPUART1** ((*uint8_t*)0x04)
- #define: **GPIO_AF4_TIM22** ((*uint8_t*)0x04)
- #define: **GPIO_AF4_EVENTOUT** ((*uint8_t*)0x04)
- #define: **GPIO_AF5_SPI2** ((*uint8_t*)0x05)
- #define: **GPIO_AF5_I2C2** ((*uint8_t*)0x05)
- #define: **GPIO_AF5_TIM2** ((*uint8_t*)0x05)
- #define: **GPIO_AF5_TIM21** ((*uint8_t*)0x05)
- #define: **GPIO_AF5_TIM22** ((*uint8_t*)0x05)
- #define: **GPIO_AF6_I2C2** ((*uint8_t*)0x06)
- #define: **GPIO_AF6_TIM21** ((*uint8_t*)0x06)
- #define: **GPIO_AF6_EVENTOUT** ((*uint8_t*)0x06)

- #define: **GPIO_AF7_COMP1** ((*uint8_t*)0x07)
- #define: **GPIO_AF7_COMP2** ((*uint8_t*)0x07)

19 HAL I2C Generic Driver

19.1 I2C Firmware driver introduction

19.2 I2C Firmware driver registers structures

19.2.1 I2C_HandleTypeDef

I2C_HandleTypeDef is defined in the `stm32l0xx_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*

19.2.2 I2C_InitTypeDef

I2C_InitTypeDef is defined in the `stm32l0xx_hal_i2c.h`

Data Fields

- `uint32_t Timing`
- `uint32_t OwnAddress1`
- `uint32_t AddressingMode`
- `uint32_t DualAddressMode`
- `uint32_t OwnAddress2`
- `uint32_t OwnAddress2Masks`
- `uint32_t GeneralCallMode`
- `uint32_t NoStretchMode`

Field Documentation

- **`uint32_t I2C_InitTypeDef::Timing`**
 - Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- **`uint32_t I2C_InitTypeDef::OwnAddress1`**
 - Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- **`uint32_t I2C_InitTypeDef::AddressingMode`**
 - Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [*I2C_addressing_mode*](#)
- **`uint32_t I2C_InitTypeDef::DualAddressMode`**
 - Specifies if dual addressing mode is selected. This parameter can be a value of [*I2C_dual_addressing_mode*](#)
- **`uint32_t I2C_InitTypeDef::OwnAddress2`**
 - Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- **`uint32_t I2C_InitTypeDef::OwnAddress2Masks`**
 - Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of [*I2C_own_address2_masks*](#)
- **`uint32_t I2C_InitTypeDef::GeneralCallMode`**
 - Specifies if general call mode is selected. This parameter can be a value of [*I2C_general_call_addressing_mode*](#)
- **`uint32_t I2C_InitTypeDef::NoStretchMode`**
 - Specifies if nostretch mode is selected. This parameter can be a value of [*I2C_nostretch_mode*](#)

19.2.3 I2C_TypeDef

I2C_TypeDef is defined in the `stm32l051xx.h`

Data Fields

- `__IO uint32_t CR1`

- `__IO uint32_t CR2`
- `__IO uint32_t OAR1`
- `__IO uint32_t OAR2`
- `__IO uint32_t TIMINGR`
- `__IO uint32_t TIMEOUTR`
- `__IO uint32_t ISR`
- `__IO uint32_t ICR`
- `__IO uint32_t PECR`
- `__IO uint32_t RXDR`
- `__IO uint32_t TXDR`

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 1 register, Address offset: 0x08
- `__IO uint32_t I2C_TypeDef::OAR2`
 - I2C Own address 2 register, Address offset: 0x0C
- `__IO uint32_t I2C_TypeDef::TIMINGR`
 - I2C Timing register, Address offset: 0x10
- `__IO uint32_t I2C_TypeDef::TIMEOUTR`
 - I2C Timeout register, Address offset: 0x14
- `__IO uint32_t I2C_TypeDef::ISR`
 - I2C Interrupt and status register, Address offset: 0x18
- `__IO uint32_t I2C_TypeDef::ICR`
 - I2C Interrupt clear register, Address offset: 0x1C
- `__IO uint32_t I2C_TypeDef::PECR`
 - I2C PEC register, Address offset: 0x20
- `__IO uint32_t I2C_TypeDef::RXDR`
 - I2C Receive data register, Address offset: 0x24
- `__IO uint32_t I2C_TypeDef::TXDR`
 - I2C Transmit data register, Address offset: 0x28

19.3 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

19.3.1 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 initialized DMA handle to the hi2c DMA Tx or Rx handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream
- 3. Configure the Communication Clock Timing, Own Address1, Master Addressing Mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
- 4. Initialize the I2C registers by calling the HAL_I2C_Init() API: (+++) These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized 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 mode of operations 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_MasterTxCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCallback

- 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 : Clears the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enables the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disables the specified I2C interrupt



You can refer to the I2C HAL driver header file for more useful macros

19.3.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
 - Clock Timing
 - Own Address 1
 - Addressing mode (Master, Slave)
 - Dual Addressing mode
 - Own Address 2
 - Own Address 2 Mask
 - General call mode
 - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.
- ***HAL_I2C_Init()***
- ***HAL_I2C_DeInit()***

- *HAL_I2C_MspInit()*
- *HAL_I2C_MspDeInit()*

19.3.3 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 No_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()*

19.3.4 Peripheral State and Errors functions

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

- *HAL_I2C_GetState()*
- *HAL_I2C_GetError()*

19.3.4.1 HAL_I2C_Init

Function Name	HAL_StatusTypeDef HAL_I2C_Init (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	Initializes the I2C according to the specified parameters in the <i>I2C_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.2 HAL_I2C_DeInit

Function Name	HAL_StatusTypeDef HAL_I2C_DelInit (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	Deinitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.3 HAL_I2C_MspInit

Function Name	void HAL_I2C_MspInit (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	I2C MSP Init.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.4 HAL_I2C_MspDelInit

Function Name	void HAL_I2C_MspDelInit (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	I2C MSP DelInit.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.5 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • pData : Pointer to data buffer • Size : Amount of data to be sent • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.6 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • pData : Pointer to data buffer • Size : Amount of data to be sent • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.7 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that

	contains the configuration information for the specified I2C.
• pData : Pointer to data buffer	
• Size : Amount of data to be sent	
• Timeout : Timeout duration	
Return values	• HAL status
Notes	• None.

19.3.4.8 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	<ul style="list-style-type: none"> • hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData : Pointer to data buffer • Size : Amount of data to be sent • Timeout : Timeout duration
Return values	• HAL status
Notes	• None.

19.3.4.9 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	<ul style="list-style-type: none"> • hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	• HAL status
Notes	• None.

19.3.4.10 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	• HAL status
Notes	• None.

19.3.4.11 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	• HAL status
Notes	• None.

19.3.4.12 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	• HAL status
Notes	• None.

19.3.4.13 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	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	• HAL status
Notes	• None.

19.3.4.14 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	<ul style="list-style-type: none"> hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. DevAddress : Target device address pData : Pointer to data buffer Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

19.3.4.15 HAL_I2C_Slave_Transmit_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</code>
Function Description	Transmit in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. pData : Pointer to data buffer Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

19.3.4.16 HAL_I2C_Slave_Receive_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. pData : Pointer to data buffer Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

19.3.4.17 HAL_I2C_Mem_Write

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Write an amount of data in blocking mode to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • MemAddress : Internal memory address • MemAddSize : Size of internal memory address • pData : Pointer to data buffer • Size : Amount of data to be sent • Timeout : Timeout duration
Return values	• HAL status
Notes	• None.

19.3.4.18 HAL_I2C_Mem_Read

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Read an amount of data in blocking mode from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • MemAddress : Internal memory address • MemAddSize : Size of internal memory address • pData : Pointer to data buffer • Size : Amount of data to be sent • Timeout : Timeout duration
Return values	• HAL status
Notes	• None.

19.3.4.19 HAL_I2C_Mem_Write_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Write an amount of data in no-blocking mode with Interrupt to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • MemAddress : Internal memory address • MemAddSize : Size of internal memory address • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.20 HAL_I2C_Mem_Read_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Read an amount of data in no-blocking mode with Interrupt from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • MemAddress : Internal memory address • MemAddSize : Size of internal memory address • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.21 HAL_I2C_Mem_Write_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Write an amount of data in no-blocking mode with DMA to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • MemAddress : Internal memory address • MemAddSize : Size of internal memory address • pData : Pointer to data buffer • Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.22 HAL_I2C_Mem_Read_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Reads an amount of data in no-blocking mode with DMA from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • MemAddress : Internal memory address • MemAddSize : Size of internal memory address • pData : Pointer to data buffer • Size : Amount of data to be read
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

19.3.4.23 HAL_I2C_IsDeviceReady

Function Name	<code>HAL_StatusTypeDef HAL_I2C_IsDeviceReady (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t</code> <code>Trials, uint32_t Timeout)</code>
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress : Target device address • Trials : Number of trials • Timeout : Timeout duration
Return values	• HAL status
Notes	<ul style="list-style-type: none"> • This function is used with Memory devices

19.3.4.24 HAL_I2C_EV_IRQHandler

Function Name	<code>void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)</code>
Function Description	This function handles I2C event interrupt request.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.25 HAL_I2C_ER_IRQHandler

Function Name	<code>void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)</code>
Function Description	This function handles I2C error interrupt request.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that

contains the configuration information for the specified I2C.

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

19.3.4.26 HAL_I2C_MasterTxCpltCallback

Function Name	void HAL_I2C_MasterTxCpltCallback (<i>I2C_HandleTypeDef</i> * hi2c)
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hi2c : : Pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

19.3.4.27 HAL_I2C_MasterRxCpltCallback

Function Name	void HAL_I2C_MasterRxCpltCallback (<i>I2C_HandleTypeDef</i> * hi2c)
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hi2c : : Pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

19.3.4.28 HAL_I2C_SlaveTxCpltCallback

Function Name	void HAL_I2C_SlaveTxCpltCallback (<i>I2C_HandleTypeDef</i> * hi2c)
---------------	--

Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.29 HAL_I2C_SlaveRxCpltCallback

Function Name	void HAL_I2C_SlaveRxCpltCallback (<i>I2C_HandleTypeDef</i> * hi2c)
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.30 HAL_I2C_MemTxCpltCallback

Function Name	void HAL_I2C_MemTxCpltCallback (<i>I2C_HandleTypeDef</i> * hi2c)
Function Description	Memory Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.31 HAL_I2C_MemRxCpltCallback

Function Name	void HAL_I2C_MemRxCpltCallback (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	Memory Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.32 HAL_I2C_ErrorCallback

Function Name	void HAL_I2C_ErrorCallback (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	I2C error callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c : : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

19.3.4.33 HAL_I2C_GetState

Function Name	HAL_I2C_StateTypeDef HAL_I2C_GetState (<i>I2C_HandleTypeDef</i> * <i>hi2c</i>)
Function Description	Returns the I2C state.
Parameters	<ul style="list-style-type: none"> • hi2c : : I2C handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

19.3.4.34 HAL_I2C_GetError

Function Name	<code>uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)</code>
Function Description	Return the I2C error code.
Parameters	<ul style="list-style-type: none"> • hi2c : : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • I2C Error Code
Notes	<ul style="list-style-type: none"> • None.

19.4 I2C Firmware driver defines

19.4.1 I2C

I2C

I2C_addressing_mode

- #define: `I2C_ADDRESSINGMODE_7BIT ((uint32_t)0x00000001)`
- #define: `I2C_ADDRESSINGMODE_10BIT ((uint32_t)0x00000002)`

I2C_dual_addressing_mode

- #define: `I2C_DUALADDRESS_DISABLED ((uint32_t)0x00000000)`
- #define: `I2C_DUALADDRESS_ENABLED I2C_OAR2_OA2EN`

I2C_Flag_definition

- #define: `I2C_FLAG_TXE I2C_ISR_TXE`
- #define: `I2C_FLAG_RXNE I2C_ISR_RXNE`
- #define: `I2C_FLAG_TXIS I2C_ISR_TXIS`

-
- #define: *I2C_FLAG_ADDR I2C_ISR_ADDR*
 - #define: *I2C_FLAG_AF I2C_ISR_NACKF*
 - #define: *I2C_FLAG_STOPF I2C_ISR_STOPF*
 - #define: *I2C_FLAG_TC I2C_ISR_TC*
 - #define: *I2C_FLAG_TCR I2C_ISR_TCR*
 - #define: *I2C_FLAG_BERR I2C_ISR_BERR*
 - #define: *I2C_FLAG_ARLO I2C_ISR_ARLO*
 - #define: *I2C_FLAG_OVR I2C_ISR_OVR*
 - #define: *I2C_FLAG_PECERR I2C_ISR_PECERR*
 - #define: *I2C_FLAG_TIMEOUT I2C_ISR_TIMEOUT*
 - #define: *I2C_FLAG_ALERT I2C_ISR_ALERT*
 - #define: *I2C_FLAG_BUSY I2C_ISR_BUSY*

-
- #define: *I2C_FLAG_DIR I2C_ISR_DIR*

I2C_general_call_addressing_mode

- #define: *I2C_GENERALCALL_DISABLED ((uint32_t)0x00000000)*
- #define: *I2C_GENERALCALL_ENABLED I2C_CR1_GCEN*

I2C Interrupt configuration definition

- #define: *I2C_IT_ERRI I2C_CR1_ERRIE*
- #define: *I2C_IT_TCI I2C_CR1_TCIE*
- #define: *I2C_IT_STOPI I2C_CR1_STOPIE*
- #define: *I2C_IT_NACKI I2C_CR1_NACKIE*
- #define: *I2C_IT_ADDRI I2C_CR1_ADDRIE*
- #define: *I2C_IT_RXI I2C_CR1_RXIE*
- #define: *I2C_IT_TXI I2C_CR1_TXIE*

I2C_Memory_Address_Size

- #define: *I2C_MEMADD_SIZE_8BIT ((uint32_t)0x00000001)*

-
- #define: *I2C_MEMADD_SIZE_16BIT* ((*uint32_t*)0x00000002)

I2C_nostretch_mode

- #define: *I2C_NOSTRETCH_DISABLED* ((*uint32_t*)0x00000000)
- #define: *I2C_NOSTRETCH_ENABLED I2C_CR1_NOSTRETCH*

I2C_own_address2_masks

- #define: *I2C_OA2_NOMASK* ((*uint8_t*)0x00)
- #define: *I2C_OA2_MASK01* ((*uint8_t*)0x01)
- #define: *I2C_OA2_MASK02* ((*uint8_t*)0x02)
- #define: *I2C_OA2_MASK03* ((*uint8_t*)0x03)
- #define: *I2C_OA2_MASK04* ((*uint8_t*)0x04)
- #define: *I2C_OA2_MASK05* ((*uint8_t*)0x05)
- #define: *I2C_OA2_MASK06* ((*uint8_t*)0x06)
- #define: *I2C_OA2_MASK07* ((*uint8_t*)0x07)

I2C_ReloadEndMode_definition

-
- #define: ***I2C_RELOAD_MODE I2C_CR2_RELOAD***
 - #define: ***I2C_AUTOEND_MODE I2C_CR2_AUTOEND***
 - #define: ***I2C_SOFTEND_MODE ((uint32_t)0x00000000)***

I2C_StartStopMode_definition

- #define: ***I2C_NO_STARTSTOP ((uint32_t)0x00000000)***
- #define: ***I2C_GENERATE_STOP I2C_CR2_STOP***
- #define: ***I2C_GENERATE_START_READ (uint32_t)(I2C_CR2_START | I2C_CR2_RD_WRN)***
- #define: ***I2C_GENERATE_START_WRITE I2C_CR2_START***

20 HAL I2C Extension Driver

20.1 I2CEEx Firmware driver introduction

20.2 I2CEEx Firmware driver API description

The following section lists the various functions of the I2CEEx library.

20.2.1 Peripheral Control functions

This section provides functions allowing to:

- Configure Noise Filters
- [***HAL_I2CEEx_AnalogFilter_Config\(\)***](#)
- [***HAL_I2CEEx_DigitalFilter_Config\(\)***](#)
- [***HAL_I2CEEx_EnableWakeUp\(\)***](#)
- [***HAL_I2CEEx_DisableWakeUp\(\)***](#)

20.2.1.1 [***HAL_I2CEEx_AnalogFilter_Config***](#)

Function Name	<i>HAL_StatusTypeDef HAL_I2CEEx_AnalogFilter_Config (</i> <i>I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</i>
Function Description	Configures I2C Analog noise filter.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

20.2.1.2 [***HAL_I2CEEx_DigitalFilter_Config***](#)

Function Name	<i>HAL_StatusTypeDef HAL_I2CEEx_DigitalFilter_Config (</i> <i>I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)</i>
Function Description	Configures I2C Digital noise filter.
Parameters	<ul style="list-style-type: none"> • hi2c : : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx

	peripheral.
	<ul style="list-style-type: none"> • DigitalFilter : : Coefficient of digital noise filter between 0x00 and 0x0F.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

20.2.1.3 HAL_I2CEx_EnableWakeUp

Function Name	HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)
Function Description	Enables I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none"> • hi2c : : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

20.2.1.4 HAL_I2CEx_DisableWakeUp

Function Name	HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)
Function Description	Disables I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none"> • hi2c : : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

20.3 I2CEx Firmware driver defines

20.3.1 I2CEx

I2CEx

I2CEx_Analog_Filter

- #define: *I2C_ANALOGFILTER_ENABLED ((uint32_t)0x00000000)*

- #define: *I2C_ANALOGFILTER_DISABLED I2C_CR1_ANFOFF*

21 HAL I2S Generic Driver

21.1 I2S Firmware driver introduction

21.2 I2S Firmware driver registers structures

21.2.1 I2S_HandleTypeDef

I2S_HandleTypeDef is defined in the `stm32l0xx_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`

21.2.2 I2S_InitTypeDef

I2S_InitTypeDef is defined in the `stm32l0xx_hal_i2s.h`

Data Fields

- *uint32_t Mode*
- *uint32_t Standard*
- *uint32_t DataFormat*
- *uint32_t MCLKOutput*
- *uint32_t AudioFreq*
- *uint32_t CPOL*

Field Documentation

- *uint32_t I2S_InitTypeDef::Mode*
 - Specifies the I2S operating mode. This parameter can be a value of [I2S_Mode](#)
- *uint32_t I2S_InitTypeDef::Standard*
 - Specifies the standard used for the I2S communication. This parameter can be a value of [I2S_Standard](#)
- *uint32_t I2S_InitTypeDef::DataFormat*
 - Specifies the data format for the I2S communication. This parameter can be a value of [I2S_Data_Format](#)
- *uint32_t I2S_InitTypeDef::MCLKOutput*
 - Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of [I2S_MCLK_Output](#)
- *uint32_t I2S_InitTypeDef::AudioFreq*
 - Specifies the frequency selected for the I2S communication. This parameter can be a value of [I2S_Audio_Frequency](#)
- *uint32_t I2S_InitTypeDef::CPOL*
 - Specifies the idle state of the I2S clock. This parameter can be a value of [I2S_Clock_Polarity](#)

21.2.3 SPI_TypeDef

SPI_TypeDef is defined in the *stm32l051xx.h*

Data Fields

- *__IO uint16_t CR1*
- *uint16_t RESERVED0*
- *__IO uint16_t CR2*
- *uint16_t RESERVED1*
- *__IO uint16_t SR*
- *uint16_t RESERVED2*
- *__IO uint16_t DR*
- *uint16_t RESERVED3*
- *__IO uint16_t CRCPR*
- *uint16_t RESERVED4*
- *__IO uint16_t RXCRCR*
- *uint16_t RESERVED5*
- *__IO uint16_t TXCRCR*
- *uint16_t RESERVED6*
- *__IO uint16_t I2SCFGR*

-
- `uint16_t RESERVED7`
 - `_IO uint16_t I2SPR`
 - `uint16_t RESERVED8`

Field Documentation

- `_IO uint16_t SPI_TypeDef::CR1`
 - SPI Control register 1 (not used in I2S mode), Address offset: 0x00
- `uint16_t SPI_TypeDef::RESERVED0`
 - Reserved, 0x02
- `_IO uint16_t SPI_TypeDef::CR2`
 - SPI Control register 2, Address offset: 0x04
- `uint16_t SPI_TypeDef::RESERVED1`
 - Reserved, 0x06
- `_IO uint16_t SPI_TypeDef::SR`
 - SPI Status register, Address offset: 0x08
- `uint16_t SPI_TypeDef::RESERVED2`
 - Reserved, 0x0A
- `_IO uint16_t SPI_TypeDef::DR`
 - SPI data register, Address offset: 0x0C
- `uint16_t SPI_TypeDef::RESERVED3`
 - Reserved, 0x0E
- `_IO uint16_t SPI_TypeDef::CRCPR`
 - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10
- `uint16_t SPI_TypeDef::RESERVED4`
 - Reserved, 0x12
- `_IO uint16_t SPI_TypeDef::RXCRCR`
 - SPI Rx CRC register (not used in I2S mode), Address offset: 0x14
- `uint16_t SPI_TypeDef::RESERVED5`
 - Reserved, 0x16
- `_IO uint16_t SPI_TypeDef::TXCRCR`
 - SPI Tx CRC register (not used in I2S mode), Address offset: 0x18
- `uint16_t SPI_TypeDef::RESERVED6`
 - Reserved, 0x1A
- `_IO uint16_t SPI_TypeDef::I2SCFGR`
 - SPI_I2S configuration register, Address offset: 0x1C
- `uint16_t SPI_TypeDef::RESERVED7`
 - Reserved, 0x1E
- `_IO uint16_t SPI_TypeDef::I2SPR`
 - SPI_I2S prescaler register, Address offset: 0x20
- `uint16_t SPI_TypeDef::RESERVED8`
 - Reserved, 0x22

21.3 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

21.3.1 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 initialized DMA handle to the I2S DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
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.
4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of 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 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

21.3.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
 - Mode
 - Standard
 - Data Format
 - MCLK Output
 - Audio frequency
 - Polarity
- Call the function HAL_I2S_DeInit() to restore the default configuration of the selected I2Sx peripheral.
- `HAL_I2S_Init()`
- `HAL_I2S_DeInit()`
- `HAL_I2S_MspInit()`
- `HAL_I2S_MspDeInit()`

21.3.3 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 No_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()*

21.3.4 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()*

21.3.4.1 HAL_I2S_Init

Function Name	HAL_StatusTypeDef HAL_I2S_Init (<i>I2S_HandleTypeDef</i> * <i>hi2s</i>)
Function Description	Initializes the I2S according to the specified parameters in the <i>I2S_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

21.3.4.2 HAL_I2S_DelInit

Function Name	HAL_StatusTypeDef HAL_I2S_DelInit (<i>I2S_HandleTypeDef</i> * <i>hi2s</i>)
Function Description	Deinitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

21.3.4.3 HAL_I2S_MspInit

Function Name	void HAL_I2S_MspInit (<i>I2S_HandleTypeDef</i> * <i>hi2s</i>)
Function Description	I2S MSP Init.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.4 HAL_I2S_MspDelInit

Function Name	void HAL_I2S_MspDelInit (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	I2S MSP Delinit.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.5 HAL_I2S_Transmit

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit (<i>I2S_HandleTypeDef</i> * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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). • This function can use an Audio Frequency up to 44KHz when I2S Clock Source is 32MHz

21.3.4.6 HAL_I2S_Receive

Function Name	HAL_StatusTypeDef HAL_I2S_Receive (<i>I2S_HandleTypeDef</i> * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)
---------------	--

Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction. This function can use an Audio Frequency up to 44KHz when I2S Clock Source is 32MHz

21.3.4.7 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> 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). This function can use an Audio Frequency up to 32KHz when I2S Clock Source is 32MHz

21.3.4.8 HAL_I2S_Receive_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</code>
Function Description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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. • This function can use an Audio Frequency up to 32KHz when I2S Clock Source is 32MHz

21.3.4.9 HAL_I2S_Transmit_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</code>
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> 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).
-------	---

21.3.4.10 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> 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).

21.3.4.11 HAL_I2S_DMAPause

Function Name	HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)
Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

21.3.4.12 HAL_I2S_DMAResume

Function Name	HAL_StatusTypeDef HAL_I2S_DMAResume (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

21.3.4.13 HAL_I2S_DMAStop

Function Name	HAL_StatusTypeDef HAL_I2S_DMAStop (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

21.3.4.14 HAL_I2S_IRQHandler

Function Name	void HAL_I2S_IRQHandler (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none"> hi2s : pointer to a I2S_HandleTypeDef structure that

	contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

21.3.4.15 HAL_I2S_TxHalfCpltCallback

Function Name	void HAL_I2S_TxHalfCpltCallback (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.16 HAL_I2S_TxCpltCallback

Function Name	void HAL_I2S_TxCpltCallback (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.17 HAL_I2S_RxHalfCpltCallback

Function Name	void HAL_I2S_RxHalfCpltCallback (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	Rx Transfer half completed callbacks.

Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.18 HAL_I2S_RxCpltCallback

Function Name	void HAL_I2S_RxCpltCallback (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.19 HAL_I2S_ErrorCallback

Function Name	void HAL_I2S_ErrorCallback (<i>I2S_HandleTypeDef</i> * hi2s)
Function Description	I2S error callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

21.3.4.20 HAL_I2S_GetState

Function Name	HAL_I2S_StateTypeDef HAL_I2S_GetState (<i>I2S_HandleTypeDef</i> * hi2s)
---------------	---

Function Description	Return the I2S state.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

21.3.4.21 HAL_I2S_GetError

Function Name	HAL_I2S_ErrorTypeDef HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S error code.
Parameters	<ul style="list-style-type: none"> • hi2s : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • I2S Error Code
Notes	<ul style="list-style-type: none"> • None.

21.4 I2S Firmware driver defines

21.4.1 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_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 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_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)

22 HAL IRDA Generic Driver

22.1 IRDA Firmware driver introduction

22.2 IRDA Firmware driver registers structures

22.2.1 IRDA_HandleTypeDef

IRDA_HandleTypeDef is defined in the `stm32l0xx_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`
- `uint16_t Mask`
- `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`
- `uint16_t IRDA_HandleTypeDef::Mask`
- `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`

22.2.2 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the `stm32l0xx_hal_irda.h`

Data Fields

- `uint32_t BaudRate`
- `uint32_t WordLength`
- `uint32_t Parity`
- `uint16_t Mode`
- `uint8_t Prescaler`
- `uint16_t PowerMode`

Field Documentation

- `uint32_t IRDA_InitTypeDef::BaudRate`
 - This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- `uint32_t IRDA_InitTypeDef::WordLength`
 - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [*IRDAEx_Word_Length*](#)
- `uint32_t IRDA_InitTypeDef::Parity`
 - Specifies the parity mode. This parameter can be a value of [*IRDA_Parity*](#)
- `uint16_t IRDA_InitTypeDef::Mode`
 - Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [*IRDA_Mode*](#)
- `uint8_t IRDA_InitTypeDef::Prescaler`
 - Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency. Prescaler value 0 is forbidden
- `uint16_t IRDA_InitTypeDef::PowerMode`
 - Specifies the IRDA power mode. This parameter can be a value of [*IRDA_Low_Power*](#)

22.2.3 USART_TypeDef

USART_TypeDef is defined in the `stm32l051xx.h`

Data Fields

- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t CR3`
- `__IO uint32_t BRR`
- `__IO uint16_t GTPR`
- `uint16_t RESERVED2`
- `__IO uint32_t RTOR`
- `__IO uint16_t RQR`
- `uint16_t RESERVED3`
- `__IO uint32_t ISR`

-
- `__IO uint32_t ICR`
 - `__IO uint16_t RDR`
 - `uint16_t RESERVED4`
 - `__IO uint16_t TDR`
 - `uint16_t RESERVED5`

Field Documentation

- `__IO uint32_t USART_TypeDef::CR1`
 - USART Control register 1, Address offset: 0x00
- `__IO uint32_t USART_TypeDef::CR2`
 - USART Control register 2, Address offset: 0x04
- `__IO uint32_t USART_TypeDef::CR3`
 - USART Control register 3, Address offset: 0x08
- `__IO uint32_t USART_TypeDef::BRR`
 - USART Baud rate register, Address offset: 0x0C
- `__IO uint16_t USART_TypeDef::GTPR`
 - USART Guard time and prescaler register, Address offset: 0x10
- `uint16_t USART_TypeDef::RESERVED2`
 - Reserved, 0x12
- `__IO uint32_t USART_TypeDef::RTOR`
 - USART Receiver Time Out register, Address offset: 0x14
- `__IO uint16_t USART_TypeDef::RQR`
 - USART Request register, Address offset: 0x18
- `uint16_t USART_TypeDef::RESERVED3`
 - Reserved, 0x1A
- `__IO uint32_t USART_TypeDef::ISR`
 - USART Interrupt and status register, Address offset: 0x1C
- `__IO uint32_t USART_TypeDef::ICR`
 - USART Interrupt flag Clear register, Address offset: 0x20
- `__IO uint16_t USART_TypeDef::RDR`
 - USART Receive Data register, Address offset: 0x24
- `uint16_t USART_TypeDef::RESERVED4`
 - Reserved, 0x26
- `__IO uint16_t USART_TypeDef::TDR`
 - USART Transmit Data register, Address offset: 0x28
- `uint16_t USART_TypeDef::RESERVED5`
 - Reserved, 0x2A

22.3 IRDA Firmware driver API description

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

22.3.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:

- Baud Rate
- Word Length
- Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit)
- Power mode
- Prescaler setting
- Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

- [*HAL_IRDA_Init\(\)*](#)
- [*HAL_IRDA_DeInit\(\)*](#)
- [*HAL_IRDA_MspInit\(\)*](#)
- [*HAL_IRDA_MspDeInit\(\)*](#)

22.3.2 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA asynchronous 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.
 - 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 IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's are :
 - HAL_IRDA_Transmit()
 - HAL_IRDA_Receive()
3. Non-Blocking mode API's with Interrupt are :
 - HAL_IRDA_Transmit_IT()
 - HAL_IRDA_Receive_IT()
 - HAL_IRDA_IRQHandler()
 - IRDA_Transmit_IT()
 - IRDA_Receive_IT()
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 No_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()*

22.3.3 Peripheral Control functions

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

- *HAL_IRDA_GetState()* API can be helpful to check in run-time the state of the IRDA peripheral.
- *IRDA_SetConfig()* API is used to configure the IRDA communications parameters.
- *HAL_IRDA_GetState()*
- *HAL_IRDA_GetError()*

22.3.3.1 HAL_IRDA_Init

Function Name	HAL_StatusTypeDef HAL_IRDA_Init (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	Initializes the IRDA mode according to the specified parameters in the <i>IRDA_InitTypeDef</i> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

22.3.3.2 HAL_IRDA_DelInit

Function Name	HAL_StatusTypeDef HAL_IRDA_DelInit (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	Deinitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

22.3.3.3 HAL_IRDA_MspInit

Function Name	void HAL_IRDA_MspInit (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	IRDA MSP Init.
Parameters	<ul style="list-style-type: none">• hirda : IRDA handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

22.3.3.4 HAL_IRDA_MspDelInit

Function Name	void HAL_IRDA_MspDelInit (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	IRDA MSP DelInit.
Parameters	<ul style="list-style-type: none">• hirda : IRDA handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

22.3.3.5 HAL_IRDA_Transmit

Function Name	HAL_StatusTypeDef HAL_IRDA_Transmit (<i>IRDA_HandleTypeDef</i> * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none">• hirda : IRDA handle• pData : pointer to data buffer• Size : amount of data to be sent• Timeout : Duration of the timeout
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

22.3.3.6 HAL_IRDA_Receive

Function Name	<code>HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle • pData : pointer to data buffer • Size : amount of data to be received • Timeout : Duration of the timeout
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

22.3.3.7 HAL_IRDA_Transmit_IT

Function Name	<code>HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</code>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle • pData : pointer to data buffer • Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

22.3.3.8 HAL_IRDA_Receive_IT

Function Name	<code>HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle • pData : pointer to data buffer

-
- | | |
|---------------|--|
| Return values | <ul style="list-style-type: none"> Size : amount of data to be received HAL status |
| Notes | <ul style="list-style-type: none"> None. |

22.3.3.9 HAL_IRDA_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (<i>IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size</i>)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> hirda : IRDA handle pData : pointer to data buffer Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

22.3.3.10 HAL_IRDA_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_IRDA_Receive_DMA (<i>IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size</i>)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> hirda : IRDA handle pData : pointer to data buffer Size : amount of data to be received
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the IRDA parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

22.3.3.11 HAL_IRDA_IRQHandler

Function Name	void HAL_IRDA IRQHandler (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	This function handles IRDA interrupt request.
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

22.3.3.12 HAL_IRDA_TxCpltCallback

Function Name	void HAL_IRDA_TxCpltCallback (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hirda : irda handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

22.3.3.13 HAL_IRDA_RxCpltCallback

Function Name	void HAL_IRDA_RxCpltCallback (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hirda : irda handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

22.3.3.14 HAL_IRDA_ErrorCallback

Function Name	void HAL_IRDA_ErrorCallback (<i>IRDA_HandleTypeDef</i> * hirda)
---------------	---

	hirda)
Function Description	IRDA error callback.
Parameters	<ul style="list-style-type: none"> • hirda : IRDA handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

22.3.3.15 HAL_IRDA_GetState

Function Name	HAL_IRDA_StateTypeDef HAL_IRDA_GetState (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	return the IRDA state
Parameters	<ul style="list-style-type: none"> • hirda : irda handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

22.3.3.16 HAL_IRDA_GetError

Function Name	uint32_t HAL_IRDA_GetError (<i>IRDA_HandleTypeDef</i> * hirda)
Function Description	Return the IRDA error code.
Parameters	<ul style="list-style-type: none"> • hirda : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA.
Return values	<ul style="list-style-type: none"> • IRDA Error Code
Notes	<ul style="list-style-type: none"> • None.

22.4 IRDA Firmware driver defines

22.4.1 IRDA

IRDA

IRDA_DMA_Rx

- #define: *IRDA_DMA_RX_DISABLE* ((*uint32_t*)0x0000)
- #define: *IRDA_DMA_RX_ENABLE* ((*uint32_t*)*USART_CR3_DMAR*)

IRDA_DMA_Tx

- #define: *IRDA_DMA_TX_DISABLE* ((*uint32_t*)0x00000000)
- #define: *IRDA_DMA_TX_ENABLE* ((*uint32_t*)*USART_CR3_DMAT*)

IRDA_Flags

- #define: *IRDA_FLAG_RXACK* ((*uint32_t*)0x00400000)
- #define: *IRDA_FLAG_TEACK* ((*uint32_t*)0x00200000)
- #define: *IRDA_FLAG_BUSY* ((*uint32_t*)0x00010000)
- #define: *IRDA_FLAG_ABRF* ((*uint32_t*)0x00008000)
- #define: *IRDA_FLAG_ABRE* ((*uint32_t*)0x00004000)
- #define: *IRDA_FLAG_TXE* ((*uint32_t*)0x00000080)
- #define: *IRDA_FLAG_TC* ((*uint32_t*)0x00000040)

-
- #define: ***IRDA_FLAG_RXNE*** ((*uint32_t*)0x00000020)
 - #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_Interruption_Mask

- #define: ***IRDA_IT_MASK*** ((*uint16_t*)0x001F)

IRDA Interrupt definition

- #define: ***IRDA_IT_PE*** ((*uint16_t*)0x0028)
- #define: ***IRDA_IT_TXE*** ((*uint16_t*)0x0727)
- #define: ***IRDA_IT_TC*** ((*uint16_t*)0x0626)
- #define: ***IRDA_IT_RXNE*** ((*uint16_t*)0x0525)
- #define: ***IRDA_IT_IDLE*** ((*uint16_t*)0x0424)
- #define: ***IRDA_IT_ERR*** ((*uint16_t*)0x0060)

YYYYY : Interrupt source position in the XX register (5bits)
 XX : Interrupt source register
 (2bits) 01: CR1 register
 10: CR2 register
 11: CR3 register

- #define: ***IRDA_IT_ORE*** ((*uint16_t*)0x0300)
ZZZZ : Flag position in the ISR register(4bits)
- #define: ***IRDA_IT_NE*** ((*uint16_t*)0x0200)
- #define: ***IRDA_IT_FE*** ((*uint16_t*)0x0100)

IRDA_IT_CLEAR_Flags

- #define: ***IRDA_CLEAR_PEF USART_ICR_PECF***
Parity Error Clear Flag
- #define: ***IRDA_CLEAR_FEF USART_ICR_FECF***
Framing Error Clear Flag
- #define: ***IRDA_CLEAR_NEF USART_ICR_NCF***
Noise detected Clear Flag
- #define: ***IRDA_CLEAR_OREF USART_ICR_ORECF***
OverRun Error Clear Flag
- #define: ***IRDA_CLEAR_TCF USART_ICR_TCCF***
Transmission Complete Clear Flag

IRDA_Low_Power

- #define: ***IRDA_POWERMODE_NORMAL*** ((*uint32_t*)0x0000)
- #define: ***IRDA_POWERMODE_LOWPOWER*** ((*uint32_t*)***USART_CR3_IRLP***)

IRDA_Mode

- #define: ***IRDA_MODE_DISABLE*** ((*uint32_t*)0x0000)

- #define: *IRDA_MODE_ENABLE* ((*uint32_t*)*USART_CR3_IREN*)

IRDA_One_Bit

- #define: *IRDA_ONE_BIT_SAMPLE_DISABLED* ((*uint32_t*)0x00000000)
- #define: *IRDA_ONE_BIT_SAMPLE_ENABLED* ((*uint32_t*)*USART_CR3_ONEBIT*)

IRDA_Parity

- #define: *IRDA_PARITY_NONE* ((*uint32_t*)0x0000)
- #define: *IRDA_PARITY_EVEN* ((*uint32_t*)*USART_CR1_PCE*)
- #define: *IRDA_PARITY_ODD* ((*uint32_t*)(*USART_CR1_PCE* | *USART_CR1_PS*))

IRDA_Request_Parameters

- #define: *IRDA_AUTOBAUD_REQUEST* ((*uint16_t*)*USART_RQR_ABRRQ*)
Auto-Baud Rate Request
- #define: *IRDA_RXDATA_FLUSH_REQUEST* ((*uint16_t*)*USART_RQR_RXFRQ*)
Receive Data flush Request
- #define: *IRDA_TXDATA_FLUSH_REQUEST* ((*uint16_t*)*USART_RQR_TXFRQ*)
Transmit data flush Request

IRDA_State

- #define: *IRDA_STATE_DISABLE* ((*uint32_t*)0x0000)

-
- #define: ***IRDA_STATE_ENABLE ((uint32_t)USART_CR1_UE)***

IRDA_Transfer_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))***

23 HAL IRDA Extension Driver

23.1 IRDAEx Firmware driver defines

23.1.1 IRDAEx

IRDAEx

IRDAEx_Word_Length

- #define: *IRDA_WORDLENGTH_7B* ((*uint32_t*)*USART_CR1_M_1*)

- #define: *IRDA_WORDLENGTH_8B* ((*uint32_t*)0x00000000)

- #define: *IRDA_WORDLENGTH_9B* ((*uint32_t*)*USART_CR1_M_0*)

24 HAL IWDG Generic Driver

24.1 IWDG Firmware driver introduction

24.2 IWDG Firmware driver registers structures

24.2.1 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the `stm32l0xx_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

24.2.2 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the `stm32l0xx_hal_iwdg.h`

Data Fields

- *uint32_t Prescaler*
- *uint32_t Reload*
- *uint32_t Window*

Field Documentation

- *uint32_t IWDG_InitTypeDef::Prescaler*
 - Select the prescaler of the IWDG. This parameter can be a value of *IWDG_Prescaler*
- *uint32_t IWDG_InitTypeDef::Reload*

- Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
 - Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

24.2.3 IWDG_TypeDef

IWDG_TypeDef is defined in the stm32l051xx.h

Data Fields

- ***_IO uint32_t KR***
- ***_IO uint32_t PR***
- ***_IO uint32_t RLR***
- ***_IO uint32_t SR***
- ***_IO uint32_t WINR***

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
- ***_IO uint32_t IWDG_TypeDef::WINR***
 - IWDG Window register, Address offset: 0x10

24.3 IWDG Firmware driver API description

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

24.3.1 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 0xFFFF. 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): ~0.512ms / ~32.0s The IWDG timeout may vary due to LSI frequency dispersion. STM32L0xx 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.

24.3.2 How to use this driver

If Window option is disabled

- Use IWDG using HAL_IWDG_Init() function to :
 - Enable write access to IWDG_PR, IWDG_RLR.
 - Configure the IWDG prescaler, counter reload value. This reload value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
- Use IWDG using HAL_IWDG_Start() function to :
 - 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.

If Window option is enabled:

- Use IWDG using HAL_IWDG_Start() function to enable IWDG downcounter
- Use IWDG using HAL_IWDG_Init() function to :
 - Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
 - Configure the IWDG prescaler, reload value and window value.
- 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

24.3.3 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

- Manage Window option
- Initialize the IWDG MSP
- [*HAL_IWDG_Init\(\)*](#)
- [*HAL_IWDG_MspInit\(\)*](#)

24.3.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.
- [*HAL_IWDG_Start\(\)*](#)
- [*HAL_IWDG_Refresh\(\)*](#)

24.3.5 Peripheral State functions

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

- [*HAL_IWDG_GetState\(\)*](#)

24.3.5.1 HAL_IWDG_Init

Function Name	HAL_StatusTypeDef HAL_IWDG_Init (<i>IWDG_HandleTypeDef</i> * <i>hiwdg</i>)
Function Description	Initializes the IWDG according to the specified parameters in the <i>IWDG_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hiwdg : pointer to a <i>IWDG_HandleTypeDef</i> structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

24.3.5.2 HAL_IWDG_MspInit

Function Name	void HAL_IWDG_MspInit (<i>IWDG_HandleTypeDef</i> * <i>hiwdg</i>)
Function Description	Initializes the IWDG MSP.
Parameters	<ul style="list-style-type: none"> • hiwdg : pointer to a <i>IWDG_HandleTypeDef</i> structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • None.

Notes	<ul style="list-style-type: none"> None.
-------	---

24.3.5.3 HAL_IWDG_Start

Function Name	HAL_StatusTypeDef HAL_IWDG_Start (<i>IWDG_HandleTypeDef</i> * <i>hiwdg</i>)
Function Description	Starts the IWDG.
Parameters	<ul style="list-style-type: none"> hiwdg : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

24.3.5.4 HAL_IWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_IWDG_Refresh (<i>IWDG_HandleTypeDef</i> * <i>hiwdg</i>)
Function Description	Refreshes the IWDG.
Parameters	<ul style="list-style-type: none"> hiwdg : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

24.3.5.5 HAL_IWDG_GetState

Function Name	HAL_IWDG_StateTypeDef HAL_IWDG_GetState (<i>IWDG_HandleTypeDef</i> * <i>hiwdg</i>)
Function Description	Returns the IWDG state.

Parameters	<ul style="list-style-type: none"> • hiwdg : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

24.4 IWDG Firmware driver defines

24.4.1 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

- #define: **IWDG_FLAG_WVU** ((uint32_t)0x0004)

Watchdog counter window 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

IWDG_Window

- #define: **IWDG_WINDOW_DISABLE** 0xFFFF

25 HAL LCD Generic Driver

25.1 LCD Firmware driver introduction

25.2 LCD Firmware driver registers structures

25.2.1 LCD_HandleTypeDef

LCD_HandleTypeDef is defined in the `stm32l0xx_hal_lcd.h`

Data Fields

- *LCD_TypeDef * Instance*
- *LCD_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_LCD_StateTypeDef State*
- *__IO HAL_LCD_ErrorTypeDef ErrorCode*

Field Documentation

- *LCD_TypeDef* LCD_HandleTypeDef::Instance*
- *LCD_InitTypeDef LCD_HandleTypeDef::Init*
- *HAL_LockTypeDef LCD_HandleTypeDef::Lock*
- *__IO HAL_LCD_StateTypeDef LCD_HandleTypeDef::State*
- *__IO HAL_LCD_ErrorTypeDef LCD_HandleTypeDef::ErrorCode*

25.2.2 LCD_InitTypeDef

LCD_InitTypeDef is defined in the `stm32l0xx_hal_lcd.h`

Data Fields

- *uint32_t Prescaler*
- *uint32_t Divider*
- *uint32_t Duty*
- *uint32_t Bias*
- *uint32_t VoltageSource*
- *uint32_t Contrast*
- *uint32_t DeadTime*
- *uint32_t PulseOnDuration*
- *uint32_t HighDrive*
- *uint32_t BlinkMode*
- *uint32_t BlinkFrequency*

Field Documentation

- **`uint32_t LCD_InitTypeDef::Prescaler`**
 - Configures the LCD Prescaler. This parameter can be one value of `LCD_Prescaler`
- **`uint32_t LCD_InitTypeDef::Divider`**
 - Configures the LCD Divider. This parameter can be one value of `LCD_Divider`
- **`uint32_t LCD_InitTypeDef::Duty`**
 - Configures the LCD Duty. This parameter can be one value of `LCD_Duty`
- **`uint32_t LCD_InitTypeDef::Bias`**
 - Configures the LCD Bias. This parameter can be one value of `LCD_Bias`
- **`uint32_t LCD_InitTypeDef::VoltageSource`**
 - Selects the LCD Voltage source. This parameter can be one value of `LCD_Voltage_Source`
- **`uint32_t LCD_InitTypeDef::Contrast`**
 - Configures the LCD Contrast. This parameter can be one value of `LCD_Contrast`
- **`uint32_t LCD_InitTypeDef::DeadTime`**
 - Configures the LCD Dead Time. This parameter can be one value of `LCD_DeadTime`
- **`uint32_t LCD_InitTypeDef::PulseOnDuration`**
 - Configures the LCD Pulse On Duration. This parameter can be one value of `LCD_PulseOnDuration`
- **`uint32_t LCD_InitTypeDef::HighDrive`**
 - Enable or disable the low resistance divider. This parameter can be set to ENABLE or DISABLE.
- **`uint32_t LCD_InitTypeDef::BlinkMode`**
 - Configures the LCD Blink Mode. This parameter can be one value of `LCD_BlinkMode`
- **`uint32_t LCD_InitTypeDef::BlinkFrequency`**
 - Configures the LCD Blink frequency. This parameter can be one value of `LCD_BlinkFrequency`

25.2.3 LCD_TypeDef

`LCD_TypeDef` is defined in the `stm32l053xx.h`

Data Fields

- `__IO uint32_t CR`
- `__IO uint32_t FCR`
- `__IO uint32_t SR`
- `__IO uint32_t CLR`
- `uint32_t RESERVED`
- `__IO uint32_t RAM`

Field Documentation

- `__IO uint32_t LCD_TypeDef::CR`
 - LCD control register, Address offset: 0x00

- `__IO uint32_t LCD_TypeDef::FCR`
 - LCD frame control register, Address offset: 0x04
- `__IO uint32_t LCD_TypeDef::SR`
 - LCD status register, Address offset: 0x08
- `__IO uint32_t LCD_TypeDef::CLR`
 - LCD clear register, Address offset: 0x0C
- `uint32_t LCD_TypeDef::RESERVED`
 - Reserved, Address offset: 0x10
- `__IO uint32_t LCD_TypeDef::RAM`
 - LCD display memory, Address offset: 0x14-0x50

25.3 LCD Firmware driver API description

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

25.3.1 How to use this driver

The LCD HAL driver can be used as follows:

1. Declare a LCD_HandleTypeDef handle structure.
2. Initialize the LCD low level resources by implementing the HAL_LCD_MspInit() API:
 - a. Enable the LCDCLK (same as RTCCLK): to configure the RTCCLK/LCDCLK, 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.
 - Select the RTC clock source using the `__HAL_RCC_RTC_CONFIG()` function. The frequency generator allows you to achieve various LCD frame rates starting from an LCD input clock frequency (LCDCLK) which can vary from 32 kHz up to 1 MHz.
 - b. LCD pins configuration:
 - Enable the clock for the LCD GPIOs.
 - Configure these LCD pins as alternate function no-pull.
 - c. Enable the LCD interface clock.
3. Program the Prescaler, Divider, Blink mode, Blink Frequency Duty, Bias, Voltage Source, Dead Time, Pulse On Duration and Contrast in the hlcd Init structure.
4. Initialize the LCD registers by calling the HAL_LCD_Init() API. The HAL_LCD_Init() API configures also the low level Hardware GPIO, CLOCK, ...etc) by calling the custumed HAL_LCD_MspInit() API. After calling the HAL_LCD_Init() the LCD RAM memory is cleared
5. Optionally you can update the LCD configuration using these macros:
 - LCD High Drive using the `__HAL_LCD_HIGHDRIVER_ENABLE()` and `__HAL_LCD_HIGHDRIVER_DISABLE()` macros
 - LCD Pulse ON Duration using the `__HAL_LCD_PULSEONDURATION_CONFIG()` macro
 - LCD Dead Time using the `__HAL_LCD_DEADTIME_CONFIG()` macro
 - The LCD Blink mode and frequency using the `__HAL_LCD_BLINK_CONFIG()` macro
 - The LCD Contrast using the `__HAL_LCD_CONTRAST_CONFIG()` macro

-
6. Write to the LCD RAM memory using the HAL_LCD_Write() API, this API can be called more time to update the different LCD RAM registers before calling HAL_LCD_UpdateDisplayRequest() API.
 7. The HAL_LCD_Clear() API can be used to clear the LCD RAM memory.
 8. When LCD RAM memory is updated, enable the update display request using the HAL_LCD_UpdateDisplayRequest() API.

LCD and low power modes:

1. The LCD remain active during STOP mode.

25.3.2 Initialization and Configuration functions

- [*HAL_LCD_DelInit\(\)*](#)
- [*HAL_LCD_Init\(\)*](#)
- [*HAL_LCD_MspDelInit\(\)*](#)
- [*HAL_LCD_MspInit\(\)*](#)

25.3.3 IO operation functions

Using its double buffer memory the LCD controller ensures the coherency of the displayed information without having to use interrupts to control LCD_RAM modification.

The application software can access the first buffer level (LCD_RAM) through the APB interface. Once it has modified the LCD_RAM using the HAL_LCD_Write() API, it sets the UDR flag in the LCD_SR register using the HAL_LCD_UpdateDisplayRequest() API.

This UDR flag (update display request) requests the updated information to be moved into the second buffer level (LCD_DISPLAY).

This operation is done synchronously with the frame (at the beginning of the next frame), until the update is completed, the LCD_RAM is write protected and the UDR flag stays high. Once the update is completed another flag (UDD - Update Display Done) is set and generates an interrupt if the UDDIE bit in the LCD_FCR register is set.

The time it takes to update LCD_DISPLAY is, in the worst case, one odd and one even frame. The update will not occur (UDR = 1 and UDD = 0) until the display is enabled (LCDEN = 1).

- [*HAL_LCD_Write\(\)*](#)
- [*HAL_LCD_Clear\(\)*](#)
- [*HAL_LCD_UpdateDisplayRequest\(\)*](#)

25.3.4 Peripheral State functions

This subsection provides a set of functions allowing to control the LCD:

- HAL_LCD_GetState() API can be helpful to check in run-time the state of the LCD peripheral State.
- HAL_LCD_GetError() API to return the LCD error code.
- [*HAL_LCD_GetState\(\)*](#)
- [*HAL_LCD_GetError\(\)*](#)

25.3.4.1 HAL_LCD_DelInit

Function Name	HAL_StatusTypeDef HAL_LCD_DelInit (LCD_HandleTypeDef * hlcd)
Function Description	Deinitializes the LCD peripheral.
Parameters	<ul style="list-style-type: none">• hlcd : LCD handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

25.3.4.2 HAL_LCD_Init

Function Name	HAL_StatusTypeDef HAL_LCD_Init (LCD_HandleTypeDef * hlcd)
Function Description	Initializes the LCD peripheral according to the specified parameters in the LCD_InitStruct.
Parameters	<ul style="list-style-type: none">• hlcd : LCD handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• This function can be used only when the LCD is disabled.

25.3.4.3 HAL_LCD_MspDelInit

Function Name	void HAL_LCD_MspDelInit (LCD_HandleTypeDef * hlcd)
Function Description	LCD MSP DelInit.
Parameters	<ul style="list-style-type: none">• hlcd : LCD handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

25.3.4.4 HAL_LCD_MspInit

Function Name	void HAL_LCD_MspInit (<i>LCD_HandleTypeDef</i> * hlcd)
Function Description	LCD MSP Init.
Parameters	<ul style="list-style-type: none"> • hlcd : LCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

25.3.4.5 HAL_LCD_Write

Function Name	HAL_StatusTypeDef HAL_LCD_Write (<i>LCD_HandleTypeDef</i> * hlcd, uint32_t RAMRegisterIndex, uint32_t RAMRegisterMask, uint32_t Data)
Function Description	Writes a word in the specific LCD RAM.
Parameters	<ul style="list-style-type: none"> • hlcd : LCD handle • RAMRegisterIndex : specifies the LCD RAM Register. This parameter can be one of the following values: <ul style="list-style-type: none"> – LCD_RAM_REGISTER0 : LCD RAM Register 0 – LCD_RAM_REGISTER1 : LCD RAM Register 1 – LCD_RAM_REGISTER2 : LCD RAM Register 2 – LCD_RAM_REGISTER3 : LCD RAM Register 3 – LCD_RAM_REGISTER4 : LCD RAM Register 4 – LCD_RAM_REGISTER5 : LCD RAM Register 5 – LCD_RAM_REGISTER6 : LCD RAM Register 6 – LCD_RAM_REGISTER7 : LCD RAM Register 7 – LCD_RAM_REGISTER8 : LCD RAM Register 8 – LCD_RAM_REGISTER9 : LCD RAM Register 9 – LCD_RAM_REGISTER10 : LCD RAM Register 10 – LCD_RAM_REGISTER11 : LCD RAM Register 11 – LCD_RAM_REGISTER12 : LCD RAM Register 12 – LCD_RAM_REGISTER13 : LCD RAM Register 13 – LCD_RAM_REGISTER14 : LCD RAM Register 14 – LCD_RAM_REGISTER15 : LCD RAM Register 15 • RAMRegisterMask : specifies the LCD RAM Register Data Mask. • Data : specifies LCD Data Value to be written.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

25.3.4.6 HAL_LCD_Clear

Function Name	HAL_StatusTypeDef HAL_LCD_Clear (LCD_HandleTypeDef * hlcd)
Function Description	Clears the LCD RAM registers.
Parameters	<ul style="list-style-type: none"> • hlcd : LCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

25.3.4.7 HAL_LCD_UpdateDisplayRequest

Function Name	HAL_StatusTypeDef HAL_LCD_UpdateDisplayRequest (LCD_HandleTypeDef * hlcd)
Function Description	Enables the Update Display Request.
Parameters	<ul style="list-style-type: none"> • hlcd : LCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • Each time software modifies the LCD_RAM it must set the UDR bit to transfer the updated data to the second level buffer. The UDR bit stays set until the end of the update and during this time the LCD_RAM is write protected. • When the display is disabled, the update is performed for all LCD_DISPLAY locations. When the display is enabled, the update is performed only for locations for which commons are active (depending on DUTY). For example if DUTY = 1/2, only the LCD_DISPLAY of COM0 and COM1 will be updated.

25.3.4.8 HAL_LCD_GetState

Function Name	HAL_LCD_StateTypeDef HAL_LCD_GetState (LCD_HandleTypeDef * hlcd)
---------------	--

Function Description	Returns the LCD state.
Parameters	<ul style="list-style-type: none"> • hlcd : LCD handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

25.3.4.9 HAL_LCD_GetError

Function Name	<code>uint32_t HAL_LCD_GetError (LCD_HandleTypeDef * hlcd)</code>
Function Description	Return the LCD error code.
Parameters	<ul style="list-style-type: none"> • hlcd : LCD handle
Return values	<ul style="list-style-type: none"> • LCD Error Code
Notes	<ul style="list-style-type: none"> • None.

25.4 LCD Firmware driver defines

25.4.1 LCD

LCD

LCD_Bias

- #define: `LCD_BIAS_1_4 ((uint32_t)0x00000000)`

1/4 Bias

- #define: `LCD_BIAS_1_2 LCD_CR_BIAS_0`

1/2 Bias

- #define: `LCD_BIAS_1_3 LCD_CR_BIAS_1`

1/3 Bias

LCD_BlinkFrequency

- #define: `LCD_BLINKFREQUENCY_DIV8 ((uint32_t)0x00000000)`

The Blink frequency = fLCD/8

- #define: **LCD_BLINKFREQUENCY_DIV16** ((uint32_t)0x00002000)

The Blink frequency = fLCD/16

- #define: **LCD_BLINKFREQUENCY_DIV32** ((uint32_t)0x00004000)

The Blink frequency = fLCD/32

- #define: **LCD_BLINKFREQUENCY_DIV64** ((uint32_t)0x00006000)

The Blink frequency = fLCD/64

- #define: **LCD_BLINKFREQUENCY_DIV128** ((uint32_t)0x00008000)

The Blink frequency = fLCD/128

- #define: **LCD_BLINKFREQUENCY_DIV256** ((uint32_t)0x0000A000)

The Blink frequency = fLCD/256

- #define: **LCD_BLINKFREQUENCY_DIV512** ((uint32_t)0x0000C000)

The Blink frequency = fLCD/512

- #define: **LCD_BLINKFREQUENCY_DIV1024** ((uint32_t)0x0000E000)

The Blink frequency = fLCD/1024

LCD_BlinkMode

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

Blink disabled

- #define: **LCD_BLINKMODE_SEGO_COM0** ((uint32_t)0x00010000)

Blink enabled on SEG[0], COM[0] (1 pixel)

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

Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)

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

Blink enabled on all SEG and all COM (all pixels)

LCD_Contrast

- #define: **LCD_CONTRASTLEVEL_0** ((uint32_t)0x00000000)

Maximum Voltage = 2.60V

- #define: **LCD_CONTRASTLEVEL_1** ((*uint32_t*)0x00000400)

Maximum Voltage = 2.73V

- #define: **LCD_CONTRASTLEVEL_2** ((*uint32_t*)0x00000800)

Maximum Voltage = 2.86V

- #define: **LCD_CONTRASTLEVEL_3** ((*uint32_t*)0x00000C00)

Maximum Voltage = 2.99V

- #define: **LCD_CONTRASTLEVEL_4** ((*uint32_t*)0x00001000)

Maximum Voltage = 3.12V

- #define: **LCD_CONTRASTLEVEL_5** ((*uint32_t*)0x00001400)

Maximum Voltage = 3.25V

- #define: **LCD_CONTRASTLEVEL_6** ((*uint32_t*)0x00001800)

Maximum Voltage = 3.38V

- #define: **LCD_CONTRASTLEVEL_7** ((*uint32_t*)0x00001C00)

Maximum Voltage = 3.51V

LCD_DeadTime

- #define: **LCD_DEADTIME_0** ((*uint32_t*)0x00000000)

No dead Time

- #define: **LCD_DEADTIME_1** ((*uint32_t*)0x00000080)

One Phase between different couple of Frame

- #define: **LCD_DEADTIME_2** ((*uint32_t*)0x00000100)

Two Phase between different couple of Frame

- #define: **LCD_DEADTIME_3** ((*uint32_t*)0x00000180)

Three Phase between different couple of Frame

- #define: **LCD_DEADTIME_4** ((uint32_t)0x00000200)

Four Phase between different couple of Frame

- #define: **LCD_DEADTIME_5** ((uint32_t)0x00000280)

Five Phase between different couple of Frame

- #define: **LCD_DEADTIME_6** ((uint32_t)0x00000300)

Six Phase between different couple of Frame

- #define: **LCD_DEADTIME_7** ((uint32_t)0x00000380)

Seven Phase between different couple of Frame

LCD_Divider

- #define: **LCD_DIVIDER_16** ((uint32_t)0x00000000)

LCD frequency = CLKPS/16

- #define: **LCD_DIVIDER_17** ((uint32_t)0x00040000)

LCD frequency = CLKPS/17

- #define: **LCD_DIVIDER_18** ((uint32_t)0x00080000)

LCD frequency = CLKPS/18

- #define: **LCD_DIVIDER_19** ((uint32_t)0x000C0000)

LCD frequency = CLKPS/19

- #define: **LCD_DIVIDER_20** ((uint32_t)0x00100000)

LCD frequency = CLKPS/20

- #define: **LCD_DIVIDER_21** ((uint32_t)0x00140000)

LCD frequency = CLKPS/21

- #define: **LCD_DIVIDER_22** ((uint32_t)0x00180000)

LCD frequency = CLKPS/22

- #define: **LCD_DIVIDER_23** ((uint32_t)0x001C0000)

LCD frequency = CLKPS/23

- #define: **LCD_DIVIDER_24** ((*uint32_t*)0x00200000)
LCD frequency = CLKPS/24

- #define: **LCD_DIVIDER_25** ((*uint32_t*)0x00240000)
LCD frequency = CLKPS/25

- #define: **LCD_DIVIDER_26** ((*uint32_t*)0x00280000)
LCD frequency = CLKPS/26

- #define: **LCD_DIVIDER_27** ((*uint32_t*)0x002C0000)
LCD frequency = CLKPS/27

- #define: **LCD_DIVIDER_28** ((*uint32_t*)0x00300000)
LCD frequency = CLKPS/28

- #define: **LCD_DIVIDER_29** ((*uint32_t*)0x00340000)
LCD frequency = CLKPS/29

- #define: **LCD_DIVIDER_30** ((*uint32_t*)0x00380000)
LCD frequency = CLKPS/30

- #define: **LCD_DIVIDER_31** ((*uint32_t*)0x003C0000)
LCD frequency = CLKPS/31

LCD_Duty

- #define: **LCD_DUTY_STATIC** ((*uint32_t*)0x00000000)
Static duty

- #define: **LCD_DUTY_1_2** ((*uint32_t*)0x00000004)
1/2 duty

- #define: **LCD_DUTY_1_3** ((*uint32_t*)0x00000008)
1/3 duty

- #define: **LCD_DUTY_1_4** ((*uint32_t*)0x0000000C)

1/4 duty

- #define: **LCD_DUTY_1_8** ((*uint32_t*)0x00000010)

1/4 duty

LCD_Flag

- #define: **LCD_FLAG_ENS LCD_SR_ENS**

- #define: **LCD_FLAG_SOF LCD_SR_SOF**

- #define: **LCD_FLAG_UDR LCD_SR_UDR**

- #define: **LCD_FLAG_UDD LCD_SR_UDD**

- #define: **LCD_FLAG_RDY LCD_SR_RDY**

- #define: **LCD_FLAG_FCRSF LCD_SR_FCRSR**

LCD Interrupts

- #define: **LCD_IT_SOF LCD_FCR_SOFIE**

- #define: **LCD_IT_UDD LCD_FCR_UDDIE**

LCD_Prescaler

- #define: **LCD_PRESCALER_1** ((*uint32_t*)0x00000000)

CLKPS = LCDCLK

- #define: **LCD_PRESCALER_2** ((*uint32_t*)0x00400000)

$CLKPS = LCDCLK/2$

- #define: **LCD_PRESCALER_4** ((*uint32_t*)0x00800000)

$CLKPS = LCDCLK/4$

- #define: **LCD_PRESCALER_8** ((*uint32_t*)0x00C00000)

$CLKPS = LCDCLK/8$

- #define: **LCD_PRESCALER_16** ((*uint32_t*)0x01000000)

$CLKPS = LCDCLK/16$

- #define: **LCD_PRESCALER_32** ((*uint32_t*)0x01400000)

$CLKPS = LCDCLK/32$

- #define: **LCD_PRESCALER_64** ((*uint32_t*)0x01800000)

$CLKPS = LCDCLK/64$

- #define: **LCD_PRESCALER_128** ((*uint32_t*)0x01C00000)

$CLKPS = LCDCLK/128$

- #define: **LCD_PRESCALER_256** ((*uint32_t*)0x02000000)

$CLKPS = LCDCLK/256$

- #define: **LCD_PRESCALER_512** ((*uint32_t*)0x02400000)

$CLKPS = LCDCLK/512$

- #define: **LCD_PRESCALER_1024** ((*uint32_t*)0x02800000)

$CLKPS = LCDCLK/1024$

- #define: **LCD_PRESCALER_2048** ((*uint32_t*)0x02C00000)

$CLKPS = LCDCLK/2048$

- #define: **LCD_PRESCALER_4096** ((*uint32_t*)0x03000000)

$CLKPS = LCDCLK/4096$

- #define: **LCD_PRESCALER_8192** ((*uint32_t*)0x03400000)

$CLKPS = LCDCLK/8192$

- #define: **LCD_PRESALER_16384** ((*uint32_t*)0x03800000)
 $CLKPS = LCDCLK/16384$

- #define: **LCD_PRESALER_32768** ((*uint32_t*)0x03C00000)
 $CLKPS = LCDCLK/32768$

LCD_PulseOnDuration

- #define: **LCD_PULSEONDURATION_0** ((*uint32_t*)0x00000000)

Pulse ON duration = 0 pulse

- #define: **LCD_PULSEONDURATION_1** ((*uint32_t*)0x00000010)

Pulse ON duration = 1/CK_PS

- #define: **LCD_PULSEONDURATION_2** ((*uint32_t*)0x00000020)

Pulse ON duration = 2/CK_PS

- #define: **LCD_PULSEONDURATION_3** ((*uint32_t*)0x00000030)

Pulse ON duration = 3/CK_PS

- #define: **LCD_PULSEONDURATION_4** ((*uint32_t*)0x00000040)

Pulse ON duration = 4/CK_PS

- #define: **LCD_PULSEONDURATION_5** ((*uint32_t*)0x00000050)

Pulse ON duration = 5/CK_PS

- #define: **LCD_PULSEONDURATION_6** ((*uint32_t*)0x00000060)

Pulse ON duration = 6/CK_PS

- #define: **LCD_PULSEONDURATION_7** ((*uint32_t*)0x00000070)

Pulse ON duration = 7/CK_PS

LCD_RAMRegister

- #define: **LCD_RAM_REGISTER0** ((*uint32_t*)0x00000000)

LCD RAM Register 0

- #define: **LCD_RAM_REGISTER1** ((*uint32_t*)0x00000001)
LCD RAM Register 1

- #define: **LCD_RAM_REGISTER2** ((*uint32_t*)0x00000002)
LCD RAM Register 2

- #define: **LCD_RAM_REGISTER3** ((*uint32_t*)0x00000003)
LCD RAM Register 3

- #define: **LCD_RAM_REGISTER4** ((*uint32_t*)0x00000004)
LCD RAM Register 4

- #define: **LCD_RAM_REGISTER5** ((*uint32_t*)0x00000005)
LCD RAM Register 5

- #define: **LCD_RAM_REGISTER6** ((*uint32_t*)0x00000006)
LCD RAM Register 6

- #define: **LCD_RAM_REGISTER7** ((*uint32_t*)0x00000007)
LCD RAM Register 7

- #define: **LCD_RAM_REGISTER8** ((*uint32_t*)0x00000008)
LCD RAM Register 8

- #define: **LCD_RAM_REGISTER9** ((*uint32_t*)0x00000009)
LCD RAM Register 9

- #define: **LCD_RAM_REGISTER10** ((*uint32_t*)0x0000000A)
LCD RAM Register 10

- #define: **LCD_RAM_REGISTER11** ((*uint32_t*)0x0000000B)
LCD RAM Register 11

- #define: **LCD_RAM_REGISTER12** ((*uint32_t*)0x0000000C)
LCD RAM Register 12

- #define: **LCD_RAM_REGISTER13** ((*uint32_t*)0x0000000D)

LCD RAM Register 13

- #define: **LCD_RAM_REGISTER14** ((*uint32_t*)0x0000000E)

LCD RAM Register 14

- #define: **LCD_RAM_REGISTER15** ((*uint32_t*)0x0000000F)

LCD RAM Register 15

LCD_Voltage_Source

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

Internal voltage source for the LCD

- #define: **LCD_VOLTAGESOURCE_EXTERNAL LCD_CR_VSEL**

External voltage source for the LCD

26 HAL LPTIM Generic Driver

26.1.1 LPTIM Firmware driver introduction

26.2 LPTIM Firmware driver registers structures

26.2.1 LPTIM_ClockConfigTypeDef

LPTIM_ClockConfigTypeDef is defined in the `stm32l0xx_hal_lptim.h`

Data Fields

- *uint32_t Source*
- *uint32_t Prescaler*

Field Documentation

- *uint32_t LPTIM_ClockConfigTypeDef::Source*
 - Selects the clock source. This parameter can be a value of [*LPTIM_Clock_Source*](#)
- *uint32_t LPTIM_ClockConfigTypeDef::Prescaler*
 - Specifies the counter clock Prescaler. This parameter can be a value of [*LPTIM_Clock_Prescaler*](#)

26.2.2 LPTIM_ULPClockConfigTypeDef

LPTIM_ULPClockConfigTypeDef is defined in the `stm32l0xx_hal_lptim.h`

Data Fields

- *uint32_t Polarity*
- *uint32_t SampleTime*

Field Documentation

- *uint32_t LPTIM_ULPClockConfigTypeDef::Polarity*
 - Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of [*LPTIM_Clock_Polarity*](#)
- *uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime*
 - Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of [*LPTIM_Clock_Sample_Time*](#)

26.2.3 LPTIM_TriggerConfigTypeDef

LPTIM_TriggerConfigTypeDef is defined in the `stm32l0xx_hal_lptim.h`

Data Fields

- `uint32_t Source`
- `uint32_t ActiveEdge`
- `uint32_t SampleTime`

Field Documentation

- `uint32_t LPTIM_TriggerConfigTypeDef::Source`
 - Selects the Trigger source. This parameter can be a value of [`LPTIM_Trigger_Source`](#)
- `uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge`
 - Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [`LPTIM_External_Trigger_Polarity`](#)
- `uint32_t LPTIM_TriggerConfigTypeDef::SampleTime`
 - Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [`LPTIM_Trigger_Sample_Time`](#)

26.2.4 LPTIM_InitTypeDef

LPTIM_InitTypeDef is defined in the `stm32l0xx_hal_lptim.h`

Data Fields

- `LPTIM_ClockConfigTypeDef Clock`
- `LPTIM_ULPClockConfigTypeDef UltraLowPowerClock`
- `LPTIM_TriggerConfigTypeDef Trigger`
- `uint32_t OutputPolarity`
- `uint32_t UpdateMode`
- `uint32_t CounterSource`

Field Documentation

- `LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock`
 - Specifies the clock parameters
- `LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock`
 - Specifies the Ultra Low Power clock parameters
- `LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger`
 - Specifies the Trigger parameters
- `uint32_t LPTIM_InitTypeDef::OutputPolarity`

- Specifies the Output polarity. This parameter can be a value of [**LPTIM_Output_Polarity**](#)
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
 - Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of [**LPTIM Updating Mode**](#)
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
 - Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of [**LPTIM Counter Source**](#)

26.2.5 LPTIM_HandleTypeDef

LPTIM_HandleTypeDef is defined in the `stm32l0xx_hal_lptim.h`

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***

Field Documentation

- ***LPTIM_TypeDef* LPTIM_HandleTypeDef::Instance***
 - Register base address
- ***LPTIM_InitTypeDef LPTIM_HandleTypeDef::Init***
 - LPTIM required parameters
- ***HAL_StatusTypeDef LPTIM_HandleTypeDef::Status***
 - LPTIM peripheral status
- ***HAL_LockTypeDef LPTIM_HandleTypeDef::Lock***
 - LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef LPTIM_HandleTypeDef::State***
 - LPTIM peripheral state

26.2.6 LPTIM_TypeDef

LPTIM_TypeDef is defined in the `stm32l051xx.h`

Data Fields

- ***__IO uint32_t ISR***
- ***__IO uint32_t ICR***
- ***__IO uint32_t IER***
- ***__IO uint32_t CFGR***
- ***__IO uint32_t CR***
- ***__IO uint32_t CMP***
- ***__IO uint32_t ARR***

-
- `_IO uint32_t CNT`

Field Documentation

- `_IO uint32_t LPTIM_TypeDef::ISR`
– LPTIM Interrupt and Status register, Address offset: 0x00
- `_IO uint32_t LPTIM_TypeDef::ICR`
– LPTIM Interrupt Clear register, Address offset: 0x04
- `_IO uint32_t LPTIM_TypeDef::IER`
– LPTIM Interrupt Enable register, Address offset: 0x08
- `_IO uint32_t LPTIM_TypeDef::CFG`
– LPTIM Configuration register, Address offset: 0x0C
- `_IO uint32_t LPTIM_TypeDef::CR`
– LPTIM Control register, Address offset: 0x10
- `_IO uint32_t LPTIM_TypeDef::CMP`
– LPTIM Compare register, Address offset: 0x14
- `_IO uint32_t LPTIM_TypeDef::ARR`
– LPTIM Autoreload register, Address offset: 0x18
- `_IO uint32_t LPTIM_TypeDef::CNT`
– LPTIM Counter register, Address offset: 0x1C

26.3 LPTIM Firmware driver API description

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

26.3.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the `LPTIM_InitTypeDef` and creates the associated handle.
- Deinitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- Deinitialize LPTIM MSP.
- [`HAL_LPTIM_Init\(\)`](#)
- [`HAL_LPTIM_DeInit\(\)`](#)
- [`HAL_LPTIM_MspInit\(\)`](#)
- [`HAL_LPTIM_MspDeInit\(\)`](#)

26.3.2 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.

- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.
- `HAL_LPTIM_PWM_Start()`
- `HAL_LPTIM_PWM_Stop()`
- `HAL_LPTIM_PWM_Start_IT()`
- `HAL_LPTIM_PWM_Stop_IT()`
- `HAL_LPTIM_OnePulse_Start()`
- `HAL_LPTIM_OnePulse_Stop()`
- `HAL_LPTIM_OnePulse_Start_IT()`
- `HAL_LPTIM_OnePulse_Stop_IT()`
- `HAL_LPTIM_SetOnce_Start()`
- `HAL_LPTIM_SetOnce_Stop()`
- `HAL_LPTIM_SetOnce_Start_IT()`
- `HAL_LPTIM_SetOnce_Stop_IT()`
- `HAL_LPTIM_Encoder_Start()`
- `HAL_LPTIM_Encoder_Stop()`
- `HAL_LPTIM_Encoder_Start_IT()`
- `HAL_LPTIM_Encoder_Stop_IT()`
- `HAL_LPTIM_TimeOut_Start()`
- `HAL_LPTIM_TimeOut_Stop()`
- `HAL_LPTIM_TimeOut_Start_IT()`
- `HAL_LPTIM_TimeOut_Stop_IT()`
- `HAL_LPTIM_Counter_Start()`
- `HAL_LPTIM_Counter_Stop()`
- `HAL_LPTIM_Counter_Start_IT()`
- `HAL_LPTIM_Counter_Stop_IT()`

26.3.3 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare) value.
- `HAL_LPTIM_ReadCounter()`
- `HAL_LPTIM_ReadAutoReload()`
- `HAL_LPTIM_ReadCompare()`

26.3.4 LPTIM IRQ handler

This section provides LPTIM IRQ handler function.

- `HAL_LPTIM_IRQHandler()`
- `HAL_LPTIM_CompareMatchCallback()`
- `HAL_LPTIM_AutoReloadMatchCallback()`
- `HAL_LPTIM_TriggerCallback()`
- `HAL_LPTIM_CompareWriteCallback()`
- `HAL_LPTIM_AutoReloadWriteCallback()`

- *HAL_LPTIM_DirectionUpCallback()*
- *HAL_LPTIM_DirectionDownCallback()*

26.3.5 Peripheral State functions

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

- *HAL_LPTIM_GetState()*

26.3.5.1 HAL_LPTIM_Init

Function Name	HAL_StatusTypeDef HAL_LPTIM_Init (<i>LPTIM_HandleTypeDef</i> * <i>hlptim</i>)
Function Description	Initializes the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hlptim : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.2 HAL_LPTIM_DeInit

Function Name	HAL_StatusTypeDef HAL_LPTIM_DeInit (<i>LPTIM_HandleTypeDef</i> * <i>hlptim</i>)
Function Description	Deinitializes the LPTIM peripheral.
Parameters	<ul style="list-style-type: none"> • hlptim : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.3 HAL_LPTIM_MspInit

Function Name	void HAL_LPTIM_MspInit (<i>LPTIM_HandleTypeDef</i> * <i>hlptim</i>)
Function Description	Initializes the LPTIM MSP.

Parameters	<ul style="list-style-type: none"> • hlptim : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.4 HAL_LPTIM_MspDeInit

Function Name	void HAL_LPTIM_MspDeInit (<i>LPTIM_HandleTypeDef</i> * hlptim)
Function Description	DeInitializes LPTIM MSP.
Parameters	<ul style="list-style-type: none"> • hlptim : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.5 HAL_LPTIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Start (<i>LPTIM_HandleTypeDef</i> * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM PWM generation.
Parameters	<ul style="list-style-type: none"> • hlptim : LPTIM handle • Period : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.6 HAL_LPTIM_PWM_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the LPTIM PWM generation.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.7 HAL_LPTIM_PWM_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM PWM generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF • Pulse : : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.8 HAL_LPTIM_PWM_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the LPTIM PWM generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.9 HAL_LPTIM_OnePulse_Start

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (</code> <code>LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t</code> <code>Pulse)</code>
Function Description	Starts the LPTIM One pulse generation.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse : : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.10 HAL_LPTIM_OnePulse_Stop

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (</code> <code>LPTIM_HandleTypeDef * hltim)</code>
Function Description	Stops the LPTIM One pulse generation.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.11 HAL_LPTIM_OnePulse_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (</code> <code>LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t</code> <code>Pulse)</code>
Function Description	Starts the LPTIM One pulse generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse : : Specifies the compare value. This parameter must

be a value between 0x0000 and 0xFFFF.

Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

26.3.5.12 HAL_LPTIM_OnePulse_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the LPTIM One pulse generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

26.3.5.13 HAL_LPTIM_SetOnce_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM in Set once mode.
Parameters	<ul style="list-style-type: none"> hltim : : LPTIM handle Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. Pulse : : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

26.3.5.14 HAL_LPTIM_SetOnce_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the LPTIM Set once mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.15 HAL_LPTIM_SetOnce_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM Set once mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse : : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.16 HAL_LPTIM_SetOnce_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the LPTIM Set once mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.17 HAL_LPTIM_Encoder_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Encoder interface.
Parameters	<ul style="list-style-type: none">• hltim : LPTIM handle• Period : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	HAL status
Notes	<ul style="list-style-type: none">• None.

26.3.5.18 HAL_LPTIM_Encoder_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Encoder interface.
Parameters	<ul style="list-style-type: none">• hltim : LPTIM handle
Return values	HAL status

26.3.5.19 HAL_LPTIM_Encoder_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Encoder interface in interrupt mode.
Parameters	<ul style="list-style-type: none">• hltim : LPTIM handle• Period : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	HAL status
Notes	<ul style="list-style-type: none">• None.

26.3.5.20 HAL_LPTIM_Encoder_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Encoder interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.21 HAL_LPTIM_TimeOut_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Timeout)
Function Description	Starts the Timeout function.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Timeout : : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.22 HAL_LPTIM_TimeOut_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Timeout function.

Parameters	<ul style="list-style-type: none"> • hlptim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.23 HAL_LPTIM_TimeOut_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)
Function Description	Starts the Timeout function in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hlptim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Timeout : : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.24 HAL_LPTIM_TimeOut_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Timeout function in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hlptim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.25 HAL_LPTIM_Counter_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Counter mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.26 HAL_LPTIM_Counter_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Counter mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.27 HAL_LPTIM_Counter_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Counter mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle • Period : : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

26.3.5.28 HAL_LPTIM_Counter_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (</code> <i>LPTIM_HandleTypeDef * hltim)</i>
Function Description	Stops the Counter mode in interrupt mode.
Parameters	<ul style="list-style-type: none">• hltim : LPTIM handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

26.3.5.29 HAL_LPTIM_ReadCounter

Function Name	<code>uint32_t HAL_LPTIM_ReadCounter (</code> <i>LPTIM_HandleTypeDef * hltim)</i>
Function Description	This function returns the current counter value.
Parameters	<ul style="list-style-type: none">• hltim : LPTIM handle
Return values	<ul style="list-style-type: none">• Counter value.
Notes	<ul style="list-style-type: none">• None.

26.3.5.30 HAL_LPTIM_ReadAutoReload

Function Name	<code>uint32_t HAL_LPTIM_ReadAutoReload (</code> <i>LPTIM_HandleTypeDef * hltim)</i>
Function Description	This function return the current Autoreload (Period) value.
Parameters	<ul style="list-style-type: none">• hltim : LPTIM handle
Return values	<ul style="list-style-type: none">• Autoreload value.

26.3.5.31 HAL_LPTIM_ReadCompare

Function Name	<code>uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hltim)</code>
Function Description	This function return the current Compare (Pulse) value.
Parameters	<ul style="list-style-type: none"> • hltim : LPTIM handle
Return values	<ul style="list-style-type: none"> • Compare value.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.32 HAL_LPTIM_IRQHandler

Function Name	<code>void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hltim)</code>
Function Description	This function handles LPTIM interrupt request.
Parameters	<ul style="list-style-type: none"> • hltim : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.33 HAL_LPTIM_CompareMatchCallback

Function Name	<code>void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hltim)</code>
Function Description	Compare match callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.34 HAL_LPTIM_AutoReloadMatchCallback

Function Name	void HAL_LPTIM_AutoReloadMatchCallback (<i>LPTIM_HandleTypeDef</i> * hltim)
Function Description	Autoreload match callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hltim : : LPTIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

26.3.5.35 HAL_LPTIM_TriggerCallback

Function Name	void HAL_LPTIM_TriggerCallback (<i>LPTIM_HandleTypeDef</i> * hltim)
Function Description	Trigger detected callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hltim : : LPTIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

26.3.5.36 HAL_LPTIM_CompareWriteCallback

Function Name	void HAL_LPTIM_CompareWriteCallback (<i>LPTIM_HandleTypeDef</i> * hltim)
Function Description	Compare write callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hltim : : LPTIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

26.3.5.37 HAL_LPTIM_AutoReloadWriteCallback

Function Name	void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Autoreload write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.38 HAL_LPTIM_DirectionUpCallback

Function Name	void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Direction counter changed from Down to Up callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.39 HAL_LPTIM_DirectionDownCallback

Function Name	void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Direction counter changed from Up to Down callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim : : LPTIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

26.3.5.40 HAL_LPTIM_GetState

Function Name	<code>HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (</code> <code>LPTIM_HandleTypeDef * hlptim)</code>
Function Description	Returns the LPTIM state.
Parameters	<ul style="list-style-type: none">• <code>hlptim</code> : LPTIM handle
Return values	<ul style="list-style-type: none">• HAL state
Notes	<ul style="list-style-type: none">• None.

26.4 LPTIM Firmware driver defines

26.4.1 LPTIM

LPTIM

LPTIM_Clock_Polarity

- #define: `LPTIM_CLOCKPOLARITY_RISINGEDGE ((uint32_t)0x00000000)`
- #define: `LPTIM_CLOCKPOLARITY_FALLINGEDGE LPTIM_CFGR_CKPOL_0`
- #define: `LPTIM_CLOCKPOLARITY_BOTHEDGES LPTIM_CFGR_CKPOL_1`

LPTIM_Clock_Prescaler

- #define: `LPTIM_PRESCALER_DIV1 ((uint32_t)0x000000)`
- #define: `LPTIM_PRESCALER_DIV2 LPTIM_CFGR_PRESC_0`
- #define: `LPTIM_PRESCALER_DIV4 LPTIM_CFGR_PRESC_1`

- #define: *LPTIM_PRESCALER_DIV8* ((*uint32_t*)(*LPTIM_CFGR_PRESC_0* | *LPTIM_CFGR_PRESC_1*))
- #define: *LPTIM_PRESCALER_DIV16 LPTIM_CFGR_PRESC_2*
- #define: *LPTIM_PRESCALER_DIV32* ((*uint32_t*)(*LPTIM_CFGR_PRESC_0* | *LPTIM_CFGR_PRESC_2*))
- #define: *LPTIM_PRESCALER_DIV64* ((*uint32_t*)(*LPTIM_CFGR_PRESC_1* | *LPTIM_CFGR_PRESC_2*))
- #define: *LPTIM_PRESCALER_DIV128* ((*uint32_t*)*LPTIM_CFGR_PRESC*)

LPTIM_Clock_Sample_Time

- #define: *LPTIM_CLOCKSAMPLETIME_DIRECTTRANSISTION* ((*uint32_t*)0x00000000)
- #define: *LPTIM_CLOCKSAMPLETIME_2TRANSISTIONS LPTIM_CFGR_CKFLT_0*
- #define: *LPTIM_CLOCKSAMPLETIME_4TRANSISTIONS LPTIM_CFGR_CKFLT_1*
- #define: *LPTIM_CLOCKSAMPLETIME_8TRANSISTIONS LPTIM_CFGR_CKFLT*

LPTIM_Clock_Source

- #define: *LPTIM_CLOCKSOURCE_APBCLOCK_LPOS* ((*uint32_t*)0x00)
- #define: *LPTIM_CLOCKSOURCE_ULPTIM LPTIM_CFGR_CKSEL*

LPTIM_Counter_Source

- #define: **LPTIM_COUNTERSOURCE_INTERNAL ((uint32_t)0x00000000)**
- #define: **LPTIM_COUNTERSOURCE_EXTERNAL LPTIM_CFGR_COUNTMODE**

LPTIM_External_Trigger_Polarity

- #define: **LPTIM_ACTIVEEDGE_RISING LPTIM_CFGR_TRIGEN_0**
- #define: **LPTIM_ACTIVEEDGE_FALLING LPTIM_CFGR_TRIGEN_1**
- #define: **LPTIM_ACTIVEEDGE_RISING_FALLING LPTIM_CFGR_TRIGEN**

LPTIM_Flag_Definition

- #define: **LPTIM_FLAG_DOWN LPTIM_ISR_DOWN**
- #define: **LPTIM_FLAG_UP LPTIM_ISR_UP**
- #define: **LPTIM_FLAG_ARROK LPTIM_ISR_ARROK**
- #define: **LPTIM_FLAG_CMPOK LPTIM_ISR_CMPOK**
- #define: **LPTIM_FLAG_EXTTRIG LPTIM_ISR_EXTTRIG**
- #define: **LPTIM_FLAG_ARRM LPTIM_ISR_ARRM**

- #define: **LPTIM_FLAG_CMPPM LPTIM_ISR_CMPPM**

LPTIM Interrupts Definition

- #define: **LPTIM_IT_DOWN LPTIM_IER_DOWNIE**
- #define: **LPTIM_IT_UP LPTIM_IER_UPIE**
- #define: **LPTIM_IT_ARROK LPTIM_IER_ARROKIE**
- #define: **LPTIM_IT_CMPOK LPTIM_IER_CMPOKIE**
- #define: **LPTIM_IT_EXTTRIG LPTIM_IER_EXTTRIGIE**
- #define: **LPTIM_IT_ARRM LPTIM_IER_ARRMIE**
- #define: **LPTIM_IT_CMPPM LPTIM_IER_CMPPMIE**

LPTIM Output Polarity

- #define: **LPTIM_OUTPUTPOLARITY_HIGH ((uint32_t)0x00000000)**
- #define: **LPTIM_OUTPUTPOLARITY_LOW (LPTIM_CFGR_WAVPOL)**

LPTIM Trigger Sample Time

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

- #define: **LPTIM_TRIGSAMPLETIME_2TRANSISTIONS LPTIM_CFGR_TRGFLT_0**
 - #define: **LPTIM_TRIGSAMPLETIME_4TRANSISTIONS LPTIM_CFGR_TRGFLT_1**
 - #define: **LPTIM_TRIGSAMPLETIME_8TRANSISTIONS LPTIM_CFGR_TRGFLT**
-
- LPTIM_Trigger_Source**
- #define: **LPTIM_TRIGSOURCE_SOFTWARE ((uint32_t)0x0000FFFF)**
 - #define: **LPTIM_TRIGSOURCE_0 ((uint32_t)0x00000000)**
 - #define: **LPTIM_TRIGSOURCE_1 ((uint32_t)LPTIM_CFGR_TRIGSEL_0)**
 - #define: **LPTIM_TRIGSOURCE_2 LPTIM_CFGR_TRIGSEL_1**
 - #define: **LPTIM_TRIGSOURCE_3 ((uint32_t)LPTIM_CFGR_TRIGSEL_0 | LPTIM_CFGR_TRIGSEL_1)**
 - #define: **LPTIM_TRIGSOURCE_4 LPTIM_CFGR_TRIGSEL_2**
 - #define: **LPTIM_TRIGSOURCE_6 ((uint32_t)LPTIM_CFGR_TRIGSEL_1 | LPTIM_CFGR_TRIGSEL_2)**
 - #define: **LPTIM_TRIGSOURCE_7 LPTIM_CFGR_TRIGSEL**

LPTIM Updating Mode

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

- #define: ***LPTIM_UPDATE_ENDOFPERIOD LPTIM_CFGR_PRELOAD***

27 HAL PCD Generic Driver

27.1.1 PCD Firmware driver introduction

27.2 PCD Firmware driver registers structures

27.2.1 PCD_EPTTypeDef

PCD_EPTTypeDef is defined in the `stm32l0xx_hal_pcd.h`

Data Fields

- `uint8_t num`
- `uint8_t is_in`
- `uint8_t is_stall`
- `uint8_t type`
- `uint16_t pmaaddress`
- `uint16_t pmaaddr0`
- `uint16_t pmaaddr1`
- `uint8_t doublebuffer`
- `uint32_t maxpacket`
- `uint8_t *xfer_buff`
- `uint32_t xfer_len`
- `uint32_t xfer_count`

Field Documentation

- **`uint8_t PCD_EPTTypeDef::num`**
 - Endpoint number This parameter must be a number between Min_Data = 1 and Max_Data = 15
- **`uint8_t PCD_EPTTypeDef::is_in`**
 - Endpoint direction This parameter must be a number between Min_Data = 0 and Max_Data = 1
- **`uint8_t PCD_EPTTypeDef::is_stall`**
 - Endpoint stall condition This parameter must be a number between Min_Data = 0 and Max_Data = 1
- **`uint8_t PCD_EPTTypeDef::type`**
 - Endpoint type This parameter can be any value of [`PCD_USB_EP_Type`](#)
- **`uint16_t PCD_EPTTypeDef::pmaaddress`**
 - PMA Address This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- **`uint16_t PCD_EPTTypeDef::pmaaddr0`**
 - PMA Address0 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- **`uint16_t PCD_EPTTypeDef::pmaaddr1`**
 - PMA Address1 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- **`uint8_t PCD_EPTTypeDef::doublebuffer`**

- Double buffer enable This parameter can be 0 or 1
- ***uint32_t PCD_EPTypedef::maxpacket***
 - Endpoint Max packet size This parameter must be a number between Min_Data = 0 and Max_Data = 64KB
- ***uint8_t* PCD_EPTypedef::xfer_buff***
 - Pointer to transfer buffer
- ***uint32_t PCD_EPTypedef::xfer_len***
 - Current transfer length
- ***uint32_t PCD_EPTypedef::xfer_count***
 - Partial transfer length in case of multi packet transfer

27.2.2 PCD_InitTypeDef

PCD_InitTypeDef is defined in the `stm32l0xx_hal_pcd.h`

Data Fields

- ***uint32_t dev_endpoints***
- ***uint32_t speed***
- ***uint32_t ep0_mps***
- ***uint32_t phy_iface***
- ***uint32_t Sof_enable***
- ***uint32_t low_power_enable***
- ***uint32_t lpm_enable***
- ***uint32_t battery_charging_enable***

Field Documentation

- ***uint32_t PCD_InitTypeDef::dev_endpoints***
 - Device Endpoints number. This parameter depends on the used USB core. This parameter must be a number between Min_Data = 1 and Max_Data = 15
- ***uint32_t PCD_InitTypeDef::speed***
 - USB Core speed. This parameter can be any value of [**PCD_Speed**](#)
- ***uint32_t PCD_InitTypeDef::ep0_mps***
 - Set the Endpoint 0 Max Packet size. This parameter can be any value of [**PCD_USB_EP0_MPS**](#)
- ***uint32_t PCD_InitTypeDef::phy_iface***
 - Select the used PHY interface. This parameter can be any value of [**PCD_USB_Core_PHY**](#)
- ***uint32_t PCD_InitTypeDef::Sof_enable***
 - Enable or disable the output of the SOF signal. This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_InitTypeDef::low_power_enable***
 - Enable or disable Low Power mode This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_InitTypeDef::lpm_enable***
 - Enable or disable Link Power Management. This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_InitTypeDef::battery_charging_enable***

- Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

27.2.3 PCD_HandleTypeDef

PCD_HandleTypeDef is defined in the `stm32l0xx_hal_pcd.h`

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***_IO uint8_t USB_Address***
- ***PCD_EPTTypeDef IN_ep***
- ***PCD_EPTTypeDef 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
- ***_IO uint8_t PCD_HandleTypeDef::USB_Address***
 - USB Address
- ***PCD_EPTTypeDef PCD_HandleTypeDef::IN_ep[5]***
 - IN endpoint parameters
- ***PCD_EPTTypeDef PCD_HandleTypeDef::OUT_ep[5]***
 - 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

27.2.4 USB_TypeDef

USB_TypeDef is defined in the `stm32l052xx.h`

Data Fields

- ***_IO uint16_t EP0R***
- ***_IO uint16_t RESERVED0***

- `__IO uint16_t EP1R`
- `__IO uint16_t RESERVED1`
- `__IO uint16_t EP2R`
- `__IO uint16_t RESERVED2`
- `__IO uint16_t EP3R`
- `__IO uint16_t RESERVED3`
- `__IO uint16_t EP4R`
- `__IO uint16_t RESERVED4`
- `__IO uint16_t EP5R`
- `__IO uint16_t RESERVED5`
- `__IO uint16_t EP6R`
- `__IO uint16_t RESERVED6`
- `__IO uint16_t EP7R`
- `__IO uint16_t RESERVED7`
- `__IO uint16_t CNTR`
- `__IO uint16_t RESERVED8`
- `__IO uint16_t ISTR`
- `__IO uint16_t RESERVED9`
- `__IO uint16_t FNR`
- `__IO uint16_t RESERVEDA`
- `__IO uint16_t DADDR`
- `__IO uint16_t RESERVEDB`
- `__IO uint16_t BTABLE`
- `__IO uint16_t RESERVEDC`
- `__IO uint16_t LPMCSR`
- `__IO uint16_t RESERVEDD`
- `__IO uint16_t BCDR`
- `__IO uint16_t RESERVEDE`

Field Documentation

- `__IO uint16_t USB_TypeDef::EP0R`
 - USB Endpoint 0 register, Address offset: 0x00
- `__IO uint16_t USB_TypeDef::RESERVED0`
 - Reserved
- `__IO uint16_t USB_TypeDef::EP1R`
 - USB Endpoint 1 register, Address offset: 0x04
- `__IO uint16_t USB_TypeDef::RESERVED1`
 - Reserved
- `__IO uint16_t USB_TypeDef::EP2R`
 - USB Endpoint 2 register, Address offset: 0x08
- `__IO uint16_t USB_TypeDef::RESERVED2`
 - Reserved
- `__IO uint16_t USB_TypeDef::EP3R`
 - USB Endpoint 3 register, Address offset: 0x0C
- `__IO uint16_t USB_TypeDef::RESERVED3`
 - Reserved
- `__IO uint16_t USB_TypeDef::EP4R`
 - USB Endpoint 4 register, Address offset: 0x10
- `__IO uint16_t USB_TypeDef::RESERVED4`
 - Reserved

-
- `__IO uint16_t USB_TypeDef::EP5R`
 - USB Endpoint 5 register, Address offset: 0x14
 - `__IO uint16_t USB_TypeDef::RESERVED5`
 - Reserved
 - `__IO uint16_t USB_TypeDef::EP6R`
 - USB Endpoint 6 register, Address offset: 0x18
 - `__IO uint16_t USB_TypeDef::RESERVED6`
 - Reserved
 - `__IO uint16_t USB_TypeDef::EP7R`
 - USB Endpoint 7 register, Address offset: 0x1C
 - `__IO uint16_t USB_TypeDef::RESERVED7`
 - Reserved
 - `__IO uint16_t USB_TypeDef::CNTR`
 - Control register, Address offset: 0x40
 - `__IO uint16_t USB_TypeDef::RESERVED8`
 - Reserved
 - `__IO uint16_t USB_TypeDef::ISTR`
 - Interrupt status register, Address offset: 0x44
 - `__IO uint16_t USB_TypeDef::RESERVED9`
 - Reserved
 - `__IO uint16_t USB_TypeDef::FNR`
 - Frame number register, Address offset: 0x48
 - `__IO uint16_t USB_TypeDef::RESERVEDA`
 - Reserved
 - `__IO uint16_t USB_TypeDef::DADDR`
 - Device address register, Address offset: 0x4C
 - `__IO uint16_t USB_TypeDef::RESERVEDB`
 - Reserved
 - `__IO uint16_t USB_TypeDef::BTABLE`
 - Buffer Table address register, Address offset: 0x50
 - `__IO uint16_t USB_TypeDef::RESERVEDC`
 - Reserved
 - `__IO uint16_t USB_TypeDef::LPMCSR`
 - LPM Control and Status register, Address offset: 0x54
 - `__IO uint16_t USB_TypeDef::RESERVEDD`
 - Reserved
 - `__IO uint16_t USB_TypeDef::BCDR`
 - Battery Charging detector register, Address offset: 0x58
 - `__IO uint16_t USB_TypeDef::RESERVEDE`
 - Reserved

27.3 PCD Firmware driver API description

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

27.3.1 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
– __USB_CLK_ENABLE;
 - 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();

27.3.2 Initialization and de-initialization functions

This section provides functions allowing to:

- [*HAL_PCD_Init\(\)*](#)
- [*HAL_PCD_DelInit\(\)*](#)
- [*HAL_PCD_MspInit\(\)*](#)
- [*HAL_PCD_MspDelInit\(\)*](#)

27.3.3 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\(\)*](#)

27.3.4 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_ActiveRemoteWakeup\(\)*](#)
- [*HAL_PCD_DeActiveRemoteWakeup\(\)*](#)

27.3.5 Peripheral State functions

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

- [*HAL_PCD_GetState\(\)*](#)

27.3.5.1 HAL_PCD_Init

Function Name	HAL_StatusTypeDef HAL_PCD_Init (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Initializes the PCD according to the specified parameters in the <i>PCD_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.2 HAL_PCD_DelInit

Function Name	HAL_StatusTypeDef HAL_PCD_DelInit (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Deinitializes the PCD peripheral.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- None.

27.3.5.3 HAL_PCD_MsplInit

Function Name	void HAL_PCD_MspInit (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Initializes the PCD MSP.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

27.3.5.4 HAL_PCD_MspDeInit

Function Name	void HAL_PCD_MspDeInit (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Deinitializes PCD MSP.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

27.3.5.5 HAL_PCD_Start

Function Name	HAL_StatusTypeDef HAL_PCD_Start (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Start The USB OTG Device.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.6 HAL_PCD_Stop

Function Name	HAL_StatusTypeDef HAL_PCD_Stop (<i>PCD_HandleTypeDef</i> * hpcd)
---------------	--

Function Description	Stop The USB OTG Device.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.7 HAL_PCD_IRQHandler

Function Name	void HAL_PCD_IRQHandler (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	This function handles PCD interrupt request.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.8 HAL_PCD_DataOutStageCallback

Function Name	void HAL_PCD_DataOutStageCallback (<i>PCD_HandleTypeDef</i> * hpcd, <i>uint8_t</i> epnum)
Function Description	Data out stage callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • None.

Notes

- None.

27.3.5.9 HAL_PCD_DataInStageCallback

Function Name	void HAL_PCD_DataInStageCallback (<i>PCD_HandleTypeDef</i> * hpcd, <i>uint8_t</i> epnum)
Function Description	Data IN stage callbacks.

Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

27.3.5.10 HAL_PCD_SetupStageCallback

Function Name	void HAL_PCD_SetupStageCallback (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Setup stage callback.
Parameters	<ul style="list-style-type: none"> • hpcd : ppp handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

27.3.5.11 HAL_PCD_SOFCallback

Function Name	void HAL_PCD_SOFCallback (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	USB Start Of Frame callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

27.3.5.12 HAL_PCD_ResetCallback

Function Name	void HAL_PCD_ResetCallback (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle

Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

27.3.5.13 HAL_PCD_SuspendCallback

Function Name	void HAL_PCD_SuspendCallback (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none">hpcd : PCD handle
Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

27.3.5.14 HAL_PCD_ResumeCallback

Function Name	void HAL_PCD_ResumeCallback (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Resume event callbacks.
Parameters	<ul style="list-style-type: none">hpcd : PCD handle
Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

27.3.5.15 HAL_PCD_ISOOUTIncompleteCallback

Function Name	void HAL_PCD_ISOOUTIncompleteCallback (<i>PCD_HandleTypeDef</i> * hpcd, uint8_t epienum)
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none">hpcd : PCD handle

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

27.3.5.16 HAL_PCD_ISOINIncompleteCallback

Function Name	void HAL_PCD_ISOINIncompleteCallback (<i>PCD_HandleTypeDef</i> * <i>hpcd</i>, <i>uint8_t</i> <i>epnum</i>)
Function Description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none"> <i>hpcd</i> : PCD handle
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

27.3.5.17 HAL_PCD_ConnectCallback

Function Name	void HAL_PCD_ConnectCallback (<i>PCD_HandleTypeDef</i> * <i>hpcd</i>)
Function Description	Connection event callbacks.
Parameters	<ul style="list-style-type: none"> <i>hpcd</i> : PCD handle
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

27.3.5.18 HAL_PCD_DisconnectCallback

Function Name	void HAL_PCD_DisconnectCallback (<i>PCD_HandleTypeDef</i> * <i>hpcd</i>)
Function Description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none"> <i>hpcd</i> : ppp handle

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

27.3.5.19 HAL_PCD_DevConnect

Function Name	HAL_StatusTypeDef HAL_PCD_DevConnect (<i>PCD_HandleTypeDef * hpcd)</i>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hpcd : PCD handle pData : pointer to data buffer Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

27.3.5.20 HAL_PCD_DevDisconnect

Function Name	HAL_StatusTypeDef HAL_PCD_DevDisconnect (<i>PCD_HandleTypeDef * hpcd)</i>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hpcd : PCD handle pData : pointer to data buffer Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

27.3.5.21 HAL_PCD_SetAddress

Function Name	HAL_StatusTypeDef HAL_PCD_SetAddress (
---------------	---

PCD_HandleTypeDef * hpcd, uint8_t address)

Function Description	Set the USB Device address.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • address : new device address
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.22 HAL_PCD_EP_Open

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Open (<i>PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t</i> <i>ep_mps, uint8_t ep_type)</i>
Function Description	Open and configure an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • ep_addr : endpoint address • ep_mps : endpoint max packet size • ep_type : endpoint type
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.23 HAL_PCD_EP_Close

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Close (<i>PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</i>
Function Description	Deactivate an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • ep_addr : endpoint address
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.24 HAL_PCD_EP_Receive

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_Receive (</code> <code>PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,</code> <code>uint32_t len)</code>
Function Description	Receive an amount of data.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • ep_addr : endpoint address • pBuf : pointer to the reception buffer • len : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

27.3.5.25 HAL_PCD_EP_GetRxCount

Function Name	<code>uint16_t HAL_PCD_EP_GetRxCount (</code> <code>PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</code>
Function Description	Get Received Data Size.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • ep_addr : endpoint address
Return values	<ul style="list-style-type: none"> • Data Size

Notes

• None.

27.3.5.26 HAL_PCD_EP_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_Transmit (</code> <code>PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,</code> <code>uint32_t len)</code>
Function Description	Send an amount of data.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • ep_addr : endpoint address • pBuf : pointer to the transmission buffer

Return values	<ul style="list-style-type: none"> len : amount of data to be sent
Notes	<ul style="list-style-type: none"> HAL status None.

27.3.5.27 HAL_PCD_EP_SetStall

Function Name	HAL_StatusTypeDef HAL_PCD_EP_SetStall (<i>PCD_HandleTypeDef</i> * hpcd, uint8_t ep_addr)
Function Description	Set a STALL condition over an endpoint.
Parameters	<ul style="list-style-type: none"> hpcd : PCD handle ep_addr : endpoint address
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

27.3.5.28 HAL_PCD_EP_ClrStall

Function Name	HAL_StatusTypeDef HAL_PCD_EP_ClrStall (<i>PCD_HandleTypeDef</i> * hpcd, uint8_t ep_addr)
Function Description	Clear a STALL condition over in an endpoint.
Parameters	<ul style="list-style-type: none"> hpcd : PCD handle ep_addr : endpoint address
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

27.3.5.29 HAL_PCD_EP_Flush

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Flush (<i>PCD_HandleTypeDef</i> * hpcd, uint8_t ep_addr)
---------------	---

Function Description	Flush an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle • ep_addr : endpoint address
Return values	• HAL status
Notes	• None.

27.3.5.30 HAL_PCD_ActiveRemoteWakeu

Function Name	HAL_StatusTypeDef HAL_PCD_ActiveRemoteWakeu (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	HAL_PCD_ActiveRemoteWakeu : active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • status
Notes	• None.

27.3.5.31 HAL_PCD_DeActiveRemoteWakeu

Function Name	HAL_StatusTypeDef HAL_PCD_DeActiveRemoteWakeu (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	HAL_PCD_DeActiveRemoteWakeu : de-active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> • hpcd : PCD handle
Return values	<ul style="list-style-type: none"> • status
Notes	• None.

27.3.5.32 HAL_PCD_GetState

Function Name	PCD_StateTypeDef HAL_PCD_GetState (<i>PCD_HandleTypeDef</i> * hpcd)
Function Description	Return the PCD state.
Parameters	<ul style="list-style-type: none"> • hpcd : : PCD handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

27.4 PCD Firmware driver defines

27.4.1 PCD

PCD

PCD_Exported_Macros

- #define: **USB_EXTI_LINE_WAKEUP ((uint32_t)0x00040000)**
External interrupt line 18 Connected to the USB FS EXTI Line
- #define: **__HAL_USB_EXTI_ENABLE_IT EXTI->IMR |= USB_EXTI_LINE_WAKEUP**
- #define: **__HAL_USB_EXTI_DISABLE_IT EXTI->IMR &= ~USB_EXTI_LINE_WAKEUP**

PCD_Speed

- #define: **PCD_SPEED_HIGH 0**
- #define: **PCD_SPEED_FULL 2**

PCD_USB_Core_PHY

- #define: **PCD_PHY_EMBEDDED 2**

PCD_USB_EP0_MPS

- #define: **DEPOCTL_MPS_64 0**

- #define: ***DEP0CTL_MPS_32*** 1
 - #define: ***DEP0CTL_MPS_16*** 2
 - #define: ***DEP0CTL_MPS_8*** 3
 - #define: ***PCD_EP0MPS_64 DEP0CTL_MPS_64***
 - #define: ***PCD_EP0MPS_32 DEP0CTL_MPS_32***
 - #define: ***PCD_EP0MPS_16 DEP0CTL_MPS_16***
 - #define: ***PCD_EP0MPS_08 DEP0CTL_MPS_8***
- PCD_USB_EP_Type***
- #define: ***EP_TYPE_CTRL*** 0
 - #define: ***EP_TYPE_ISOC*** 1
 - #define: ***EP_TYPE_BULK*** 2
 - #define: ***EP_TYPE_INTR*** 3

28 HAL PCD Extension Driver

28.1.1 PCDEEx Firmware driver introduction

28.2 PCDEEx Firmware driver API description

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

28.2.1 Peripheral extended features functions

- [*HAL_PCDEEx_PMACConfig\(\)*](#)

28.2.1.1 [*HAL_PCDEEx_PMACConfig*](#)

Function Name	<code>HAL_StatusTypeDef HAL_PCDEEx_PMACConfig (</code> <code>PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t</code> <code>ep_kind, uint32_t pmaaddress)</code>
Function Description	Configure PMA for EP.
Parameters	<ul style="list-style-type: none"> • pdev : : Device instance • ep_addr : endpoint address • ep_Kind : endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used • pmaaddress : EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.
Return values	<ul style="list-style-type: none"> • : status
Notes	<ul style="list-style-type: none"> • None.

28.3 PCDEEx Firmware driver defines

28.3.1 PCDEEx

PCDEEx

29 HAL PWR Generic Driver

29.1.1 PWR Firmware driver introduction

29.2 PWR Firmware driver registers structures

29.2.1 PWR_PVDTTypeDef

PWR_PVDTTypeDef is defined in the `stm32l0xx_hal_pwr.h`

Data Fields

- *uint32_t PVDLevel*
- *uint32_t Mode*

Field Documentation

- *uint32_t PWR_PVDTTypeDef::PVDLevel*
 - PVDLevel: Specifies the PVD detection level. This parameter can be a value of [*PWR_PVD_detection_level*](#)
- *uint32_t PWR_PVDTTypeDef::Mode*
 - Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [*PWR_PVD_Mode*](#)

29.2.2 PWR_TypeDef

PWR_TypeDef is defined in the `stm32l051xx.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

29.3 PWR Firmware driver API description

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

29.3.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__PWR_CLK_ENABLE()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- `HAL_PWR_DeInit()`
- `HAL_PWR_EnableBkUpAccess()`
- `HAL_PWR_DisableBkUpAccess()`

29.3.2 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).
- The PVD can use an external input analog voltage (PVD_IN) which is compared internally to VREFINT. The PVD_IN (PB7) has to be configured in Analog mode when PWR_PVDLevel_7 is selected (PLS[2:0] = 111).
- 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 are two WakeUp pins: WakeUp Pin 1 on PA.00. WakeUp Pin 2 on PC.13.

Main and Backup Regulators configuration

Low Power modes configuration

The device features 5 low-power modes:

- Low power run mode: regulator in low power mode, limited clock frequency, limited number of peripherals running.
- Sleep mode: Cortex-M0+ core stopped, peripherals kept running.
- Low power sleep mode: Cortex-M0+ core stopped, limited clock frequency, limited number of peripherals running, regulator in low power mode.
- Stop mode: All clocks are stopped, regulator running, regulator in low power mode.
- Standby mode: VCORE domain powered off

Low power run mode

To further reduce the consumption when the system is in Run mode, the regulator can be configured in low power mode. In this mode, the system frequency should not exceed MSI frequency range1. In Low power run mode, all I/O pins keep the same state as in Run mode.

- Entry:
 - VCORE in range2
 - Decrease the system frequency tonot exceed the frequency of MSI frequency range1.
 - The regulator is forced in low power mode using the HAL_PWREx_EnableLowPowerRunMode() function.
- Exit:
 - The regulator is forced in Main regulator mode using the HAL_PWREx_DisableLowPowerRunMode() function.
 - Increase the system frequency if needed.

Sleep mode

- Entry: The Sleep mode is entered by using the HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFx) functions with
 - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
 - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- Exit:
 - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

Low power sleep mode

- Entry: The Low power sleep mode is entered by using the HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFx) functions with
 - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
 - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- The Flash memory can be switched off by using the control bits (SLEEP_PD in the FLASH_ACR register. This reduces power consumption but increases the wake-up time.
- Exit:
 - If the WFI instruction was used to enter Low power sleep mode, any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Low power sleep mode. If the WFE instruction was used to enter Low power sleep mode, the MCU exits Sleep mode as soon as an event occurs.

Stop mode

The Stop mode is based on the Cortex-M0+ deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the VCORE domain are stopped, the PLL, the MSI, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. To get the lowest consumption in Stop mode, the internal Flash memory also enters low

power mode. When the Flash memory is in power-down mode, an additional startup delay is incurred when waking up from Stop mode. To minimize the consumption In Stop mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering Stop mode. They can be switched on again by software after exiting Stop mode using the ULP bit in the PWR_CR register. In Stop mode, all I/O pins keep the same state as in Run mode.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI) function with:
 - Main regulator ON.
 - Low Power regulator ON.
 - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
 - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- Exit:
 - By issuing an interrupt or a wakeup event, the MSI or HSI16 RC oscillator is selected as system clock depending the bit STOPWUCK in the RCC_CFGR register

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M0+ deepsleep mode, with the voltage regulator disabled. The VCORE domain is consequently powered off. The PLL, the MSI, 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 and Standby circuitry. To minimize the consumption In Standby mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering the Standby mode. They can be switched on again by software after exiting the Standby mode. function.

- 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, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with an RTC alarm event, it is necessary to:
 - Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
 - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function
 - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
 - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to:
 - Configure the EXTI Line 19 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
 - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.

- Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
- To wake up from the Stop mode with an RTC WakeUp event, it is necessary to:
 - Configure the EXTI Line 20 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
 - Enable the RTC WakeUp Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to generate the RTC WakeUp event using the RTC_WakeUpClockConfig(), RTC_SetWakeUpCounter() and RTC_WakeUpCmd() functions.
- RTC auto-wakeup (AWU) from the Standby mode
 - To wake up from the Standby mode with an RTC alarm event, it is necessary to:
 - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
 - To wake up from the Standby mode with an RTC Tamper or time stamp event, it is necessary to:
 - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
 - To wake up from the Standby mode with an RTC WakeUp event, it is necessary to:
 - Enable the RTC WakeUp Interrupt using the RTC_ITConfig() function
 - Configure the RTC to generate the RTC WakeUp event using the RTC_WakeUpClockConfig(), RTC_SetWakeUpCounter() and RTC_WakeUpCmd() functions.
- Comparator auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with an comparator 1 or comparator 2 wakeup event, it is necessary to:
 - Configure the EXTI Line 21 for comparator 1 or EXTI Line 22 for comparator 2 to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the EXTI_Init() function.
 - Configure the comparator to generate the event.
- [**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\(\)**](#)

29.3.2.1 HAL_PWR_DeInit

Function Name	void HAL_PWR_DeInit (void)
Function Description	Deinitializes the HAL PWR peripheral registers to their default

reset values.

Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

29.3.2.2 HAL_PWR_EnableBkUpAccess

Function Name	void HAL_PWR_EnableBkUpAccess (void)
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers).
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

29.3.2.3 HAL_PWR_DisableBkUpAccess

Function Name	void HAL_PWR_DisableBkUpAccess (void)
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers).
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

29.3.2.4 HAL_PWR_PVDConfig

Function Name	void HAL_PWR_PVDCConfig (<i>PWR_PVDTTypeDef</i> * sConfigPVD)
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> sConfigPVD : pointer to an <i>PWR_PVDTTypeDef</i> structure that contains the configuration information for the PVD.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

29.3.2.5 HAL_PWR_EnablePVD

Function Name	void HAL_PWR_EnablePVD (void)
Function Description	Enables the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

29.3.2.6 HAL_PWR_DisablePVD

Function Name	void HAL_PWR_DisablePVD (void)
Function Description	Disables the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

29.3.2.7 HAL_PWR_EnableWakeUpPin

Function Name	void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)
Function Description	Enables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> • WakeUpPinx : Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: <ul style="list-style-type: none"> – PWR_WAKEUP_PIN1 : – PWR_WAKEUP_PIN2 :
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

29.3.2.8 HAL_PWR_DisableWakeUpPin

Function Name	void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)
Function Description	Disables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> • WakeUpPinx : Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: <ul style="list-style-type: none"> – PWR_WAKEUP_PIN1 : – PWR_WAKEUP_PIN2 :
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

29.3.2.9 HAL_PWR_EnterSLEEPMode

Function Name	void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)
Function Description	Enters Sleep mode.
Parameters	<ul style="list-style-type: none"> • Regulator : Specifies the regulator state in SLEEP mode. This parameter can be one of the following values: <ul style="list-style-type: none"> – PWR_MAINREGULATOR_ON : SLEEP mode with regulator ON – PWR_LOWPOWERREGULATOR_ON : SLEEP mode with low power regulator ON • SLEEPEntry : Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up

	source. 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.

29.3.2.10 HAL_PWR_EnterSTOPMode

Function Name	void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)
Function Description	Enters Stop mode.
Parameters	<ul style="list-style-type: none"> • Regulator : Specifies the regulator state in Stop mode. This parameter can be one of the following values: <ul style="list-style-type: none"> – PWR_MAINREGULATOR_ON : Stop mode with regulator ON – PWR_LOWPOWERREGULATOR_ON : Stop mode with low power regulator ON • STOPEntry : Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values: <ul style="list-style-type: none"> – PWR_STOPENTRY_WFI : Enter Stop mode with WFI instruction – PWR_STOPENTRY_WFE : Enter Stop mode with WFE instruction
Return values	• None.
Notes	<ul style="list-style-type: none"> • 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, MSI or HSI16 RC oscillator is selected as system clock depending on the bit STOPWUCK in the RCC_CFGR register. • 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.

29.3.2.11 HAL_PWR_EnterSTANDBYMode

Function Name	void HAL_PWR_EnterSTANDBYMode (void)
Function Description	Enters Standby mode.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> 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 (PC13) if configured for tamper.WKUP pin 1 (PA0) if enabled.WKUP pin 2 (PC13) if enabled.

29.3.2.12 HAL_PWR_PVD_IRQHandler

Function Name	void HAL_PWR_PVD_IRQHandler (void)
Function Description	This function handles the PWR PVD interrupt request.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> This API should be called under the PVD_IRQHandler().

29.3.2.13 HAL_PWR_PVDCallback

Function Name	void HAL_PWR_PVDCallback (void)
Function Description	PWR PVD interrupt callback.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

29.4 PWR Firmware driver defines

29.4.1 PWR

PWR

PWR_Exported_Macro

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

External interrupt line 16 Connected to the PVD EXTI Line

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_VREFINTRDY PWR_CSR_VREFINTRDYF**

- #define: **PWR_FLAG_VOS PWR_CSR_VOSF**

- #define: **PWR_FLAG_REGLP PWR_CSR_REGLPF**

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_SLEEP_STOP_mode

- #define: **PWR_MAINREGULATOR_ON ((uint32_t)0x00000000)**
- #define: **PWR_LOWPOWERREGULATOR_ON PWR_CR_LPSDSR**

PWR_Regulator_Voltage_Scale

- #define: **PWR_REGULATOR_VOLTAGE_SCALE1 PWR_CR_VOS_0**
- #define: **PWR_REGULATOR_VOLTAGE_SCALE2 PWR_CR_VOS_1**
- #define: **PWR_REGULATOR_VOLTAGE_SCALE3 PWR_CR_VOS**

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_EWUP1**
- #define: **PWR_WAKEUP_PIN2 PWR_CSR_EWUP2**

30 HAL PWR Extension Driver

30.1.1 PWREx Firmware driver introduction

30.2 PWREx Firmware driver API description

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

30.2.1 Peripheral extended features functions

- [*HAL_PWREx_EnableFastWakeUp\(\)*](#)
- [*HAL_PWREx_DisableFastWakeUp\(\)*](#)
- [*HAL_PWREx_EnableUltraLowPower\(\)*](#)
- [*HAL_PWREx_DisableUltraLowPower\(\)*](#)
- [*HAL_PWREx_EnableLowPowerRunMode\(\)*](#)
- [*HAL_PWREx_DisableLowPowerRunMode\(\)*](#)

30.2.1.1 [*HAL_PWREx_EnableFastWakeUp*](#)

Function Name	void HAL_PWREx_EnableFastWakeUp (void)
Function Description	Enables the Fast WakeUp from Ultra Low Power mode.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • This bit works in conjunction with ULP bit. Means, when ULP = 1 and FWU = 1 :VREFINT startup time is ignored when exiting from low power mode.

30.2.1.2 [*HAL_PWREx_DisableFastWakeUp*](#)

Function Name	void HAL_PWREx_DisableFastWakeUp (void)
Function Description	Disables the Fast WakeUp from Ultra Low Power mode.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

30.2.1.3 HAL_PWREx_EnableUltraLowPower

Function Name	void HAL_PWREx_EnableUltraLowPower (void)
Function Description	Enables the Ultra Low Power mode.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

30.2.1.4 HAL_PWREx_DisableUltraLowPower

Function Name	void HAL_PWREx_DisableUltraLowPower (void)
Function Description	Disables the Ultra Low Power mode.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

30.2.1.5 HAL_PWREx_EnableLowPowerRunMode

Function Name	void HAL_PWREx_EnableLowPowerRunMode (void)
Function Description	Enters the Low Power Run mode.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• Low power run mode can only be entered when VCORE is in range 2. In addition, the dynamic voltage scaling must not be used when Low power run mode is selected. Only Stop and Sleep modes with regulator configured in Low power mode is allowed when Low power run mode is selected.• In Low power run mode, all I/O pins keep the same state as in

Run mode.

30.2.1.6 HAL_PWREx_DisableLowPowerRunMode

Function Name	void HAL_PWREx_DisableLowPowerRunMode (void)
Function Description	Exits the Low Power Run mode.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

30.3 PWREx Firmware driver defines

30.3.1 PWREx

PWREx

PWREx_Exported

- #define: **__HAL_PWR_FLASHWAKEUP_ENABLE_CLEAR_BIT(PWR->CR, PWR_CR_DSEEKOFF)**

When entering low power mode (stop or standby only), if DS_EE_KOFF and RUN_PD of FLASH_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.

- #define: **__HAL_PWR_FLASHWAKEUP_DISABLE_SET_BIT(PWR->CR, PWR_CR_DSEEKOFF)**

31 HAL RCC Generic Driver

31.1.1 RCC Firmware driver introduction

31.2 RCC Firmware driver registers structures

31.2.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the `stm32l0xx_hal_rcc.h`

Data Fields

- `uint32_t PLLState`
- `uint32_t PLLSource`
- `uint32_t PLLMUL`
- `uint32_t PLLDIV`

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::PLLMUL`
 - PLLMUL: Multiplication factor for PLL VCO output clock This parameter must of `RCC_PLLMultiplication_Factor`
- `uint32_t RCC_PLLInitTypeDef::PLLDIV`
 - PLLDIV: Division factor for main system clock (SYSCLK) This parameter must be a value of [`RCC_PLLDivider_Factor`](#)

31.2.2 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the `stm32l0xx_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**](#)

31.2.3 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the `stm32l0xx_hal_rcc.h`

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSISState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSISState***
- ***uint32_t HSI48State***
- ***uint32_t MSISState***
- ***uint32_t MSICalibrationValue***
- ***uint32_t MSIClockRange***
- ***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::HSISState***
 - 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::LSISState***
 - The new state of the LSI. This parameter can be a value of [**RCC_LSI_Config**](#)

- ***uint32_t RCC_OsclInitTypeDef::HSI48State***
 - The new state of the HSI48. This parameter can be a value of **RCC_HSI48_Config**
- ***uint32_t RCC_OsclInitTypeDef::MSIState***
 - The new state of the MSI. This parameter can be a value of **RCC_MSI_Config**
- ***uint32_t RCC_OsclInitTypeDef::MSICalibrationValue***
 - The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t RCC_OsclInitTypeDef::MSIClockRange***
 - The MSI frequency range. This parameter can be a value of **RCC_MSI_Clock_Range**
- ***RCC_PLLInitTypeDef RCC_OsclInitTypeDef::PLL***
 - PLL structure parameters

31.2.4 RCC_TypeDef

RCC_TypeDef is defined in the stm32l051xx.h

Data Fields

- ***_IO uint32_t CR***
- ***_IO uint32_t ICSCR***
- ***_IO uint32_t CRRCR***
- ***_IO uint32_t CFGR***
- ***_IO uint32_t CIER***
- ***_IO uint32_t CIFR***
- ***_IO uint32_t CICR***
- ***_IO uint32_t IOPRSTR***
- ***_IO uint32_t AHBRSTR***
- ***_IO uint32_t APB2RSTR***
- ***_IO uint32_t APB1RSTR***
- ***_IO uint32_t IOPENR***
- ***_IO uint32_t AHBENR***
- ***_IO uint32_t APB2ENR***
- ***_IO uint32_t APB1ENR***
- ***_IO uint32_t IOPSMENR***
- ***_IO uint32_t AHBSMENR***
- ***_IO uint32_t APB2SMENR***
- ***_IO uint32_t APB1SMENR***
- ***_IO uint32_t CCIPR***
- ***_IO uint32_t CSR***

Field Documentation

- ***_IO uint32_t RCC_TypeDef::CR***
 - RCC clock control register, Address offset: 0x00
- ***_IO uint32_t RCC_TypeDef::ICSCR***
 - RCC Internal clock sources calibration register, Address offset: 0x04
- ***_IO uint32_t RCC_TypeDef::CRRCR***
 - RCC Clock recovery RC register, Address offset: 0x08

- **`__IO uint32_t RCC_TypeDef::CFGReg`**
 - RCC Clock configuration register, Address offset: 0x0C
- **`__IO uint32_t RCC_TypeDef::CIER`**
 - RCC Clock interrupt enable register, Address offset: 0x10
- **`__IO uint32_t RCC_TypeDef::CIFR`**
 - RCC Clock interrupt flag register, Address offset: 0x14
- **`__IO uint32_t RCC_TypeDef::CICR`**
 - RCC Clock interrupt clear register, Address offset: 0x18
- **`__IO uint32_t RCC_TypeDef::IOPRSTR`**
 - RCC IO port reset register, Address offset: 0x1C
- **`__IO uint32_t RCC_TypeDef::AHBRSTR`**
 - RCC AHB peripheral reset register, Address offset: 0x20
- **`__IO uint32_t RCC_TypeDef::APB2RSTR`**
 - RCC APB2 peripheral reset register, Address offset: 0x24
- **`__IO uint32_t RCC_TypeDef::APB1RSTR`**
 - RCC APB1 peripheral reset register, Address offset: 0x28
- **`__IO uint32_t RCC_TypeDef::IOPENR`**
 - RCC Clock IO port enable register, Address offset: 0x2C
- **`__IO uint32_t RCC_TypeDef::AHBENR`**
 - RCC AHB peripheral clock enable register, Address offset: 0x30
- **`__IO uint32_t RCC_TypeDef::APB2ENR`**
 - RCC APB2 peripheral enable register, Address offset: 0x34
- **`__IO uint32_t RCC_TypeDef::APB1ENR`**
 - RCC APB1 peripheral enable register, Address offset: 0x38
- **`__IO uint32_t RCC_TypeDef::IOPSMENR`**
 - RCC IO port clock enable in sleep mode register, Address offset: 0x3C
- **`__IO uint32_t RCC_TypeDef::AHBSMENR`**
 - RCC AHB peripheral clock enable in sleep mode register, Address offset: 0x40
- **`__IO uint32_t RCC_TypeDef::APB2SMENR`**
 - RCC APB2 peripheral clock enable in sleep mode register, Address offset: 0x44
- **`__IO uint32_t RCC_TypeDef::APB1SMENR`**
 - RCC APB1 peripheral clock enable in sleep mode register, Address offset: 0x48
- **`__IO uint32_t RCC_TypeDef::CCIPR`**
 - RCC clock configuration register, Address offset: 0x4C
- **`__IO uint32_t RCC_TypeDef::CSR`**
 - RCC Control/status register, Address offset: 0x50

31.3 RCC Firmware driver API description

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

31.3.1 RCC specific features

After reset the device is running from MSI (2 MHz) with Flash 0 WS, all peripherals are off except internal SRAM, Flash and SW-DP.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at MSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the SW-DP 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 whose clocks are not derived from the System clock (ADC, RTC/LCD, RNG and IWDG)

31.3.2 Initialization and de-initialization functions

This section provide functions allowing to configure the internal/external clocks, PLL, CSS and MCO.

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. MSI (multi-speed internal), multispeed low power RC (65.536 KHz to 4.194 MHz) MHz used as System clock source.
3. LSI (low-speed internal), 37 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 1 to 24 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL (clocked by HSI or HSE), for System clock and USB (48 MHz).
7. CSS (Clock security system), once enable and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to MSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
8. MCO (microcontroller clock output), used to output SYSCLK, HSI, MSI, HSE, PLL, LSI or LSE clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): MSI, 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 IOPORT, AHB bus (DMA,Flash...). 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 from an external clock mapped on the I2S_CKIN pin. RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use __HAL_RCC_RTC_CONFIG() and __HAL_RCC_RTC_ENABLE() macros to configure this clock. USB FS, and RNG require a frequency equal to 48 MHz to work correctly. This clock is derived from the main PLL or HSI48 RC oscillator. IWDG clock which is always the LSI clock.
2. For the STM32L0xx devices, the maximum frequency of the SYSCLK ,HCLK, APB1 and APB2 is 32 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly:
----- | Wait states | HCLK clock frequency (MHz) | | |-----
---| | (Latency) | voltage range | voltage range | | | 1.65 V - 3.6 V | 2.0 V - 3.6 V | | |-----
-----|-----|-----| | VCORE = 1.2 V | VCORE = 1.5 V | VCORE = 1.8 V
| |-----|-----|-----|-----| | 0WS(1CPU cycle)|0 < HCLK <= 2

0 < HCLK <= 8	0 < HCLK <= 16 ----- ----- ----- ----- ----- -----
1WS(2CPU cycle) 2 < HCLK <= 4	8 < HCLK <= 16 16 < HCLK <= 32 -----

- [*HAL_RCC_DeInit\(\)*](#)
- [*HAL_RCC_OscConfig\(\)*](#)
- [*HAL_RCC_ClockConfig\(\)*](#)

31.3.3 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_GetSysClockFreq\(\)*](#)
- [*HAL_RCC_GetHCLKFreq\(\)*](#)
- [*HAL_RCC_GetPCLK1Freq\(\)*](#)
- [*HAL_RCC_GetPCLK2Freq\(\)*](#)
- [*HAL_RCC_GetOscConfig\(\)*](#)
- [*HAL_RCC_GetClockConfig\(\)*](#)
- [*HAL_RCC_NMI_IRQHandler\(\)*](#)
- [*HAL_RCC_CCSCallback\(\)*](#)

31.3.3.1 HAL_RCC_DeInit

Function Name	void HAL_RCC_DeInit (void)
Function Description	Resets the RCC clock configuration to the default reset state.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • The default reset state of the clock configuration is given below: MSI ON and used as system clock source (MSI range is not modified by this function, it keep the value configured by user application)HSI, HSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS and MCO OFFAll interrupts disabled • This function doesn't modify the configuration of the <ul style="list-style-type: none"> -Peripheral clocks -HSI48, LSI, LSE and RTC clocks

31.3.3.2 HAL_RCC_OscConfig

Function Name	HAL_StatusTypeDef HAL_RCC_OscConfig (
---------------	--

RCC_OsclInitTypeDef * RCC_OsclInitStruct)

Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OsclInitTypeDef.
Parameters	<ul style="list-style-type: none"> • RCC_OsclInitStruct : pointer to an RCC_OsclInitTypeDef structure that contains the configuration information for the RCC Oscillators.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The PLL is not disabled when used as system clock.

31.3.3.3 HAL_RCC_ClockConfig

Function Name	HAL_StatusTypeDef HAL_RCC_ClockConfig (<i>RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)</i>
Function Description	Initializes the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	<ul style="list-style-type: none"> • RCC_ClkInitStruct : pointer to an RCC_OsclInitTypeDef structure that contains the configuration information for the RCC peripheral. • FLatency : FLASH Latency, this parameter depends on System Clock Frequency
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function • The MSI 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.

31.3.3.4 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 MCO pin.
Parameters	<ul style="list-style-type: none"> • RCC_MCOx : specifies the output direction for the clock source. For STM32L0xx family this parameter can have only one value: <ul style="list-style-type: none"> – RCC_MCO1 : Clock source to output on MCO pin(PA8). – RCC_MCO2 : Clock source to output on MCO pin(PA9). • RCC_MCOsource : specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> – RCC_MCO1SOURCE_NOCLOCK : No clock selected – RCC_MCO1SOURCE_SYSCLK : System clock selected – RCC_MCO1SOURCE_HSI : HSI oscillator clock selected – RCC_MCO1SOURCE_MSI : MSI oscillator clock selected – RCC_MCO1SOURCE_HSE : HSE oscillator clock selected – RCC_MCO1SOURCE_PLLCLK : PLL clock selected – RCC_MCO1SOURCE_LSI : LSI clock selected – RCC_MCO1SOURCE_LSE : LSE clock selected – RCC_MCO1SOURCE_HSI48 : HSI48 clock selected • RCC_MCODIV : specifies the MCO DIV. This parameter can be one of the following values: <ul style="list-style-type: none"> – RCC_MCODIV_1 : no division applied to MCO clock – RCC_MCODIV_2 : division by 2 applied to MCO clock – RCC_MCODIV_4 : division by 4 applied to MCO clock – RCC_MCODIV_8 : division by 8 applied to MCO clock – RCC_MCODIV_16 : division by 16 applied to MCO clock
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • MCO pin should be configured in alternate function mode.

31.3.3.5 HAL_RCC_EnableCSS

Function Name	void HAL_RCC_EnableCSS (void)
Function Description	Enables the Clock Security System.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • 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-M0+ NMI (Non-Maskable Interrupt) exception vector.

31.3.3.6 HAL_RCC_GetSysClockFreq

Function Name	<code>uint32_t HAL_RCC_GetSysClockFreq (void)</code>
Function Description	Returns the SYSCLK frequency.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • SYSCLK frequency
Notes	<ul style="list-style-type: none"> • 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 MSI, function returns values based on MSI Value as defined by the MSI range. • 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 <code>stm32l0xx_hal_conf.h</code> 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 <code>stm32l0xx_hal_conf.h</code> file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result. • The result of this function could be not correct when using fractional value for HSE crystal. • This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters. • Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

31.3.3.7 HAL_RCC_GetHCLKFreq

Function Name	<code>uint32_t HAL_RCC_GetHCLKFreq (void)</code>
Function Description	Returns the HCLK frequency.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> HCLK frequency
Notes	<ul style="list-style-type: none"> 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

31.3.3.8 HAL_RCC_GetPCLK1Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK1Freq (void)</code>
Function Description	Returns the PCLK1 frequency.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> PCLK1 frequency
Notes	<ul style="list-style-type: none"> 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.

31.3.3.9 HAL_RCC_GetPCLK2Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK2Freq (void)</code>
Function Description	Returns the PCLK2 frequency.
Parameters	<ul style="list-style-type: none"> None.
Return values	<ul style="list-style-type: none"> PCLK2 frequency
Notes	<ul style="list-style-type: none"> 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.

31.3.3.10 HAL_RCC_GetOscConfig

Function Name	void HAL_RCC_GetOscConfig (<i>RCC_OscInitTypeDef</i> * <i>RCC_OscInitStruct</i>)
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • RCC_OscInitStruct : pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

31.3.3.11 HAL_RCC_GetClockConfig

Function Name	void HAL_RCC_GetClockConfig (<i>RCC_ClkInitTypeDef</i> * <i>RCC_ClkInitStruct</i>, uint32_t * <i>pFLatency</i>)
Function Description	Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • RCC_OscInitStruct : pointer to an RCC_ClkInitTypeDef structure that will be configured. • pFLatency : Pointer on the Flash Latency.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

31.3.3.12 HAL_RCC_NMI_IRQHandler

Function Name	void HAL_RCC_NMI_IRQHandler (void)
Function Description	This function handles the RCC CSS interrupt request.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.

Notes

- This API should be called under the NMI_Handler().

31.3.3.13 HAL_RCC_CCSCallback

Function Name	void HAL_RCC_CCSCallback (void)
Function Description	RCC Clock Security System interrupt callback.
Parameters	<ul style="list-style-type: none"> • none :
Return values	<ul style="list-style-type: none"> • none
Notes	<ul style="list-style-type: none"> • None.

31.4 RCC Firmware driver defines

31.4.1 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_PPREG1_DIV1*
- #define: *RCC_HCLK_DIV2 RCC_CFGR_PPREG1_DIV2*
- #define: *RCC_HCLK_DIV4 RCC_CFGR_PPREG1_DIV4*
- #define: *RCC_HCLK_DIV8 RCC_CFGR_PPREG1_DIV8*
- #define: *RCC_HCLK_DIV16 RCC_CFGR_PPREG1_DIV16*

RCC_BitAddress_AliasRegion

- #define: *RCC_OFFSET (RCC_BASE - PERIPH_BASE)*
- #define: *RCC_CR_OFFSET (RCC_OFFSET + 0x00)*
- #define: *RCC_CFGR_OFFSET (RCC_OFFSET + 0x08)*

- #define: **RCC_CSR_OFFSET** (**RCC_OFFSET + 0x74**)

- #define: **CR_BYTE2_ADDRESS** ((**uint32_t**)0x40023802)

- #define: **CIER_BYTE0_ADDRESS** ((**uint32_t**)(**RCC_BASE + 0x10 + 0x00**))

- #define: **LSE_TIMEOUT_VALUE LSE_STARTUP_TIMEOUT**

- #define: **DBP_TIMEOUT_VALUE** ((**uint32_t**)100)

RCC_Exported

- #define: **__DMA1_CLK_ENABLE** (**RCC->AHBENR** |= (**RCC_AHBENR_DMA1EN**))

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

- #define: **__MIF_CLK_ENABLE** (**RCC->AHBENR** |= (**RCC_AHBENR_MIFEN**))

- #define: **__CRC_CLK_ENABLE** (**RCC->AHBENR** |= (**RCC_AHBENR_CRCEN**))

- #define: **__DMA1_CLK_DISABLE** (**RCC->AHBENR** &= ~ (**RCC_AHBENR_DMA1EN**)))

- #define: **__MIF_CLK_DISABLE** (**RCC->AHBENR** &= ~ (**RCC_AHBENR_MIFEN**)))

- #define: **__CRC_CLK_DISABLE** (**RCC->AHBENR** &= ~ (**RCC_AHBENR_CRCEN**)))

-
- #define: **`__GPIOA_CLK_ENABLE (RCC->IOPENR |= (RCC_IOPENR_GPIOAEN))`**

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

- #define: **`__GPIOB_CLK_ENABLE (RCC->IOPENR |= (RCC_IOPENR_GPIOBEN))`**

- #define: **`__GPIOC_CLK_ENABLE (RCC->IOPENR |= (RCC_IOPENR_GPIOCEN))`**

- #define: **`__GPIOD_CLK_ENABLE (RCC->IOPENR |= (RCC_IOPENR_GPIODEN))`**

- #define: **`__GPIOH_CLK_ENABLE (RCC->IOPENR |= (RCC_IOPENR_GPIOHEN))`**

- #define: **`__GPIOA_CLK_DISABLE (RCC->IOPENR &= ~ (RCC_IOPENR_GPIOAEN))`**

- #define: **`__GPIOB_CLK_DISABLE (RCC->IOPENR &= ~ (RCC_IOPENR_GPIOBEN))`**

- #define: **`__GPIOC_CLK_DISABLE (RCC->IOPENR &= ~ (RCC_IOPENR_GPIOCEN))`**

- #define: **`__GPIOD_CLK_DISABLE (RCC->IOPENR &= ~ (RCC_IOPENR_GPIODEN))`**

- #define: **`__GPIOH_CLK_DISABLE (RCC->IOPENR &= ~ (RCC_IOPENR_GPIOHEN))`**

- #define: **`__WWDG_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_WWDGEN))`**

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

- #define: **PWR_CLK_ENABLE** (*RCC->APB1ENR |= (RCC_APB1ENR_PWREN)*)

- #define: **WWDG_CLK_DISABLE** (*RCC->APB1ENR &= ~ (RCC_APB1ENR_WWDGEN)*)

- #define: **PWR_CLK_DISABLE** (*RCC->APB1ENR &= ~ (RCC_APB1ENR_PWREN)*)

- #define: **SYSCFG_CLK_ENABLE** (*RCC->APB2ENR |= (RCC_APB2ENR_SYSCFGEN)*)

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

- #define: **DBGMCU_CLK_ENABLE** (*RCC->APB2ENR |= (RCC_APB2ENR_DBGMCUEN)*)

- #define: **SYSCFG_CLK_DISABLE** (*RCC->APB2ENR &= ~ (RCC_APB2ENR_SYSCFGEN)*)

- #define: **DBGMCU_CLK_DISABLE** (*RCC->APB2ENR &= ~ (RCC_APB2ENR_DBGMCUEN)*)

- #define: **AHB_FORCE_RESET** (*RCC->AHBRSTR = 0xFFFFFFFF*)

- #define: **DMA1_FORCE_RESET** (*RCC->AHBRSTR |= (RCC_AHBRSTR_DMA1RST)*)

- #define: **MIF_FORCE_RESET** (*RCC->AHBRSTR |= (RCC_AHBRSTR_MIFRST)*)

- #define: **`__CRC_FORCE_RESET (RCC->AHBRSTR |= (RCC_AHBRSTR_CRCRST))`**
- #define: **`__AHB_RELEASE_RESET (RCC->AHBRSTR = 0x00)`**
- #define: **`__CRC_RELEASE_RESET (RCC->AHBRSTR &= ~ (RCC_AHBRSTR_CRCRST))`**
- #define: **`__DMA1_RELEASE_RESET (RCC->AHBRSTR &= ~ (RCC_AHBRSTR_DMA1RST))`**
- #define: **`__MIF_RELEASE_RESET (RCC->AHBRSTR &= ~ (RCC_AHBRSTR_MIFRST))`**
- #define: **`__IOP_FORCE_RESET (RCC->IOPRSTR = 0xFFFFFFFF)`**
- #define: **`__GPIOA_FORCE_RESET (RCC->IOPRSTR |= (RCC_IOPRSTR_GPIOARST))`**
- #define: **`__GPIOB_FORCE_RESET (RCC->IOPRSTR |= (RCC_IOPRSTR_GPIOBRST))`**
- #define: **`__GPIOC_FORCE_RESET (RCC->IOPRSTR |= (RCC_IOPRSTR_GPIOCRST))`**
- #define: **`__GPIOD_FORCE_RESET (RCC->IOPRSTR |= (RCC_IOPRSTR_GPIODRST))`**

- #define: ***__GPIOH_FORCE_RESET (RCC->IOPRSTR |= (RCC_IOPRSTR_GPIOHRST))***
- #define: ***__IOP_RELEASE_RESET (RCC->IOPRSTR = 0x00)***
- #define: ***__GPIOA_RELEASE_RESET (RCC->IOPRSTR &= ~(RCC_IOPRSTR_GPIOARST))***
- #define: ***__GPIOB_RELEASE_RESET (RCC->IOPRSTR &= ~(RCC_IOPRSTR_GPIOBRST))***
- #define: ***__GPIOC_RELEASE_RESET (RCC->IOPRSTR &= ~(RCC_IOPRSTR_GPIOCRST))***
- #define: ***__GPIOD_RELEASE_RESET (RCC->IOPRSTR &= ~(RCC_IOPRSTR_GPIODRST))***
- #define: ***__GPIOH_RELEASE_RESET (RCC->IOPRSTR &= ~(RCC_IOPRSTR_GPIOHRST))***
- #define: ***__APB1_FORCE_RESET (RCC->APB1RSTR = 0xFFFFFFFF)***
- #define: ***__WWDG_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_WWDGRST))***
- #define: ***__PWR_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_PWRRST))***

-
- #define: `__APB1_RELEASE_RESET (RCC->APB1RSTR = 0x00)`
 - #define: `__WWDG_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_WWDGRST))`
 - #define: `__PWR_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_PWRRST))`
 - #define: `__APB2_FORCE_RESET (RCC->APB2RSTR = 0xFFFFFFFF)`
 - #define: `__DBGMCU_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_DBGMCURST))`
 - #define: `__SYSCFG_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_SYSCFGRST))`
 - #define: `__APB2_RELEASE_RESET (RCC->APB2RSTR = 0x00)`
 - #define: `__DBGMCU_RELEASE_RESET (RCC->APB2RSTR &= ~ (RCC_APB2RSTR_DBGMCURST))`
 - #define: `__SYSCFG_RELEASE_RESET (RCC->APB2RSTR &= ~ (RCC_APB2RSTR_SYSCFGRST))`
 - #define: `__CRC_CLK_SLEEP_ENABLE (RCC->AHBSMENR |= (RCC_AHBSMENR_CRCSMEN))`

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

- #define: `__MIF_CLK_SLEEP_ENABLE (RCC->AHBSMENR |= (RCC_AHBSMENR_MIFSMEN))`

- #define: `__SRAM_CLK_SLEEP_ENABLE (RCC->AHBSMENR |= (RCC_AHBSMENR_SRAMSMEN))`

- #define: `__DMA1_CLK_SLEEP_ENABLE (RCC->AHBSMENR |= (RCC_AHBSMENR_DMA1SMEN))`

- #define: `__CRC_CLK_SLEEP_DISABLE (RCC->AHBSMENR &= ~ (RCC_AHBSMENR_CRCSMEN))`

- #define: `__MIF_CLK_SLEEP_DISABLE (RCC->AHBSMENR &= ~ (RCC_AHBSMENR_MIFSMEN))`

- #define: `__SRAM_CLK_SLEEP_DISABLE (RCC->AHBSMENR &= ~ (RCC_AHBSMENR_SRAMSMEN))`

- #define: `__DMA1_CLK_SLEEP_DISABLE (RCC->AHBSMENR &= ~ (RCC_AHBSMENR_DMA1SMEN))`

- #define: `__GPIOA_CLK_SLEEP_ENABLE (RCC->IOPSMENR |= (RCC_IOPSMENR_GPIOASMEN))`
Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

- #define: `__GPIOB_CLK_SLEEP_ENABLE (RCC->IOPSMENR |= (RCC_IOPSMENR_GPIOBSMEN))`

- #define: `__GPIOC_CLK_SLEEP_ENABLE (RCC->IOPSMENR |= (RCC_IOPSMENR_GPIOCSMEN))`

- #define: **__GPIOD_CLK_SLEEP_ENABLE (RCC->IOPSMENR |= (RCC_IOPSMENR_GPIODSMEN))**
- #define: **__GPIOH_CLK_SLEEP_ENABLE (RCC->IOPSMENR |= (RCC_IOPSMENR_GPIOHSMEN))**
- #define: **__GPIOA_CLK_SLEEP_DISABLE (RCC->IOPSMENR &= ~(RCC_IOPSMENR_GPIOASMEN))**
- #define: **__GPIOB_CLK_SLEEP_DISABLE (RCC->IOPSMENR &= ~(RCC_IOPSMENR_GPIOBSEN))**
- #define: **__GPIOC_CLK_SLEEP_DISABLE (RCC->IOPSMENR &= ~(RCC_IOPSMENR_GPIOCSMEN))**
- #define: **__GPIOD_CLK_SLEEP_DISABLE (RCC->IOPSMENR &= ~(RCC_IOPSMENR_GPIODSMEN))**
- #define: **__GPIOH_CLK_SLEEP_DISABLE (RCC->IOPSMENR &= ~(RCC_IOPSMENR_GPIOHSMEN))**
- #define: **__WWDG_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_WWDGSMEN))**

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

- #define: **__PWR_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_PWRSMEN))**

- #define: **`__WWDG_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~(RCC_APB1SMENR_WWDGSMEN))`**

- #define: **`__PWR_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~(RCC_APB1SMENR_PWRSMEN))`**

- #define: **`__SYSCFG_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_SYSCFGSMEN))`**

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

- #define: **`__DBGMCU_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_DBGMCUSMEN))`**

- #define: **`__SYSCFG_CLK_SLEEP_DISABLE (RCC->APB2SMENR &= ~(RCC_APB2SMENR_SYSCFGSMEN))`**

- #define: **`__DBGMCU_CLK_SLEEP_DISABLE (RCC->APB2SMENR &= ~(RCC_APB2SMENR_DBGMCUSMEN))`**

- #define: **`__HAL_RCC_HSI_ENABLE SET_BIT(RCC->CR, RCC_CR_HSION)`**

The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

- #define: **`__HAL_RCC_HSI_DISABLE CLEAR_BIT(RCC->CR, RCC_CR_HSION)`**

- #define: **`__HAL_RCC_MSI_ENABLE SET_BIT(RCC->CR, RCC_CR_MSION)`**

The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from

STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

- #define: **`__HAL_RCC_MSI_DISABLE CLEAR_BIT(RCC->CR, RCC_CR_MSION)`**

- #define: **`__HAL_RCC_HSI48_ENABLE do { SET_BIT(RCC->CRRCR, RCC_CRRCR_HSI48ON); \RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN; \SYSCFG->CFGGR3 |= (SYSCFG_CFGR3_ENREF_HSI48 | SYSCFG_CFGR3_EN_VREFINT); \} while (0)`**

After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable and can be used to clock the USB. The HSI48 is stopped by hardware when entering STOP and STANDBY modes.

- #define: **`__HAL_RCC_HSI48_DISABLE do { CLEAR_BIT(RCC->CRRCR, RCC_CRRCR_HSI48ON); \SYSCFG->CFGGR3 &= (uint32_t)~((uint32_t)(SYSCFG_CFGR3_ENREF_HSI48 | SYSCFG_CFGR3_EN_VREFINT)); \} while (0)`**

- #define: **`__HAL_RCC_LSI_ENABLE SET_BIT(RCC->CSR, RCC_CSR_LSION)`**
After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

- #define: **`__HAL_RCC_LSI_DISABLE CLEAR_BIT(RCC->CSR, RCC_CSR_LSION)`**

- #define: **`__HAL_RCC_RTC_ENABLE SET_BIT(RCC->CSR, RCC_CSR_RTCEN)`**
These macros must be used only after the RTC clock source was selected.

- #define: **`__HAL_RCC_RTC_DISABLE CLEAR_BIT(RCC->CSR, RCC_CSR_RTCEN)`**

- #define: **`__HAL_RCC_GET_RTC_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CSR_RTCSEL)))`**

- #define: **`__HAL_RCC_BACKUPRESET_FORCE_SET_BIT(RCC->CSR, RCC_CSR_RTCRST)`**

This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

- #define: **`__HAL_RCC_BACKUPRESET_RELEASE_CLEAR_BIT(RCC->CSR, RCC_CSR_RTCRST)`**

- #define: **`__HAL_RCC_PLL_ENABLE_SET_BIT(RCC->CR, RCC_CR_PLLON)`**

After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

- #define: **`__HAL_RCC_PLL_DISABLE_CLEAR_BIT(RCC->CR, RCC_CR_PLLON)`**

- #define: **`__HAL_RCC_GET_SYSCLK_SOURCE ((uint32_t)(RCC->CFGR & RCC_CFGR_SWS))`**

*The clock source used as system clock. The returned value can be one of the following:
 RCC_CFGR_SWS_HSI: HSI used as system clock.
 RCC_CFGR_SWS_HSE: HSE used as system clock.
 RCC_CFGR_SWS_PLL: PLL used as system clock.*

- #define: **`__HAL_RCC_GET_PLL_OSCSOURCE ((uint32_t)(RCC->CFGR & RCC_CFGR_PLLSRC))`**

*The oscillator used as PLL clock source. The returned value can be one of the following:
 RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
 RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.*

- #define: **`__RCC_PLLSRC ((RCC->PLLCFGR & RCC_PLLCFGR_PLLSRC) >> POSITION_VAL(RCC_PLLCFGR_PLLSRC))`**

RCC_Flag

- #define: **`RCC_FLAG_HSIRDY ((uint8_t)0x22)`**

- #define: **`RCC_FLAG_HSIDIV ((uint8_t)0x24)`**

- #define: **RCC_FLAG_MSIRDY** ((*uint8_t*)0x29)
- #define: **RCC_FLAG_HSERDY** ((*uint8_t*)0x31)
- #define: **RCC_FLAG_PLLRDY** ((*uint8_t*)0x39)
- #define: **RCC_FLAG_LSERDY** ((*uint8_t*)0x49)
- #define: **RCC_FLAG_LSECSS** ((*uint8_t*)0x4E)
- #define: **RCC_FLAG_LSIRDY** ((*uint8_t*)0x41)
- #define: **RCC_FLAG_FIREWALLRST** ((*uint8_t*)0x58)
- #define: **RCC_FLAG_OBLRST** ((*uint8_t*)0x59)
- #define: **RCC_FLAG_PINRST** ((*uint8_t*)0x5A)
- #define: **RCC_FLAG_PORRST** ((*uint8_t*)0x5B)
- #define: **RCC_FLAG_SFTRST** ((*uint8_t*)0x5C)
- #define: **RCC_FLAG_IWDGRST** ((*uint8_t*)0x5D)

- #define: **RCC_FLAG_WWDGRST** ((uint8_t)0x5E)
- #define: **RCC_FLAG_LPWRST** ((uint8_t)0x5F)
- #define: **RCC_FLAG_HSI48RDY** ((uint8_t)0x61)

RCC_Flags Interrupts Management

- #define: **__HAL_RCC_CLEAR_RESET_FLAGS** (RCC->CSR |= RCC_CSR_RMVF)

The reset flags are: **RCC_FLAG_OBLRST**, **RCC_FLAG_PINRST**, **RCC_FLAG_PORRST**, **RCC_FLAG_SFTRST**, **RCC_FLAG_IWDGRST**, **RCC_FLAG_WWDGRST**, **RCC_FLAG_LPWRST**.

- #define: **RCC_FLAG_MASK** ((uint8_t)0x1F)

__FLAG : specifies the flag to check. This parameter can be one of the following values:
RCC_FLAG_HSIRDY: HSI oscillator clock ready **RCC_FLAG_MSIRDY**: MSI oscillator
 clock ready **RCC_FLAG_HSERDY**: HSE oscillator clock ready **RCC_FLAG_PLLRDY**: PLL
 clock ready **RCC_FLAG_LSECSS**: LSE oscillator clock CSS detected
RCC_FLAG_LSERDY: LSE oscillator clock ready **RCC_FLAG_LSIRDY**: LSI oscillator
 clock ready **RCC_FLAG_OBLRST**: Option Byte Loader (OBL) reset **RCC_FLAG_PINRST**:
 Pin reset **RCC_FLAG_PORRST**: POR/PDR reset **RCC_FLAG_SFTRST**: Software reset
RCC_FLAG_IWDGRST: Independent Watchdog reset **RCC_FLAG_WWDGRST**: Window
 Watchdog reset **RCC_FLAG_LPWRST**: Low Power reset The new state of __FLAG_
 (TRUE or FALSE).

RCC_HSE_Config

- #define: **RCC_HSE_OFF** ((uint32_t)0x00000000)
- #define: **RCC_HSE_ON** RCC_CR_HSEON
- #define: **RCC_HSE_BYPASS** RCC_CR_HSEBYP

RCC_HSI48_Config

- #define: **RCC_HSI48_OFF ((uint8_t)0x00)**
- #define: **RCC_HSI48_ON ((uint8_t)0x01)**

RCC_HSI_Config

- #define: **RCC_HSI_OFF ((uint8_t)0x00)**
- #define: **RCC_HSI_ON ((uint8_t)0x01)**
- #define: **RCC_HSI_DIV4 ((uint8_t)0x09)**

RCC_Interrupt

- #define: **RCC_IT_LSIRDY RCC_CIFR_LSIRDYF**
- #define: **RCC_IT_LSERDY RCC_CIFR_LSERDYF**
- #define: **RCC_IT_HSIRDY RCC_CIFR_HSIRDYF**
- #define: **RCC_IT_HSERDY RCC_CIFR_HSERDYF**
- #define: **RCC_IT_PLLRDY RCC_CIFR_PLLRDYF**
- #define: **RCC_IT_MSIRDY RCC_CIFR_MSIRDYF**
- #define: **RCC_IT_HSI48RDY RCC_CIFR_HSI48RDYF**

- #define: **RCC_IT_LSECSS RCC_CIFR_LSECSSF**
- #define: **RCC_IT_CSS RCC_CIFR_CSSF**

RCC_LSE_Config

- #define: **RCC_LSE_OFF ((uint32_t)0x00000000)**
- #define: **RCC_LSE_ON RCC_CSR_LSEON**
- #define: **RCC_LSE_BYPASS RCC_CSR_LSEBYP**

RCC_LSI_Config

- #define: **RCC_LSI_OFF ((uint8_t)0x00)**
- #define: **RCC_LSI_ON ((uint8_t)0x01)**

RCC_MCOPrescaler

- #define: **RCC_MCODIV_1 RCC_CFGR_MCO_PRE_1**
- #define: **RCC_MCODIV_2 RCC_CFGR_MCO_PRE_2**
- #define: **RCC_MCODIV_4 RCC_CFGR_MCO_PRE_4**
- #define: **RCC_MCODIV_8 RCC_CFGR_MCO_PRE_8**

- #define: **RCC_MCODIV_16 RCC_CFGR_MCO_PRE_16**

RCC_MCO_Clock_Source

- #define: **RCC_MCO1SOURCE_NOCLOCK ((uint8_t)0x00)**
- #define: **RCC_MCO1SOURCE_SYSCLK ((uint8_t)0x01)**
- #define: **RCC_MCO1SOURCE_HSI ((uint8_t)0x02)**
- #define: **RCC_MCO1SOURCE_MSI ((uint8_t)0x03)**
- #define: **RCC_MCO1SOURCE_HSE ((uint8_t)0x04)**
- #define: **RCC_MCO1SOURCE_PLLCLK ((uint8_t)0x05)**
- #define: **RCC_MCO1SOURCE_LSI ((uint8_t)0x06)**
- #define: **RCC_MCO1SOURCE_LSE ((uint8_t)0x07)**
- #define: **RCC_MCO1SOURCE_HSI48 ((uint8_t)0x08)**

RCC_MCO_Index

- #define: **RCC_MCO1 ((uint32_t)0x00000000)**

- #define: **RCC_MCO2** ((*uint32_t*)0x00000001)

RCC_MSI_Clock_Range

- #define: **RCC_MSIRANGE_0** **RCC_ICSCR_MSIRANGE_0**

MSI = 65.536 KHz

- #define: **RCC_MSIRANGE_1** **RCC_ICSCR_MSIRANGE_1**

MSI = 131.072 KHz

- #define: **RCC_MSIRANGE_2** **RCC_ICSCR_MSIRANGE_2**

MSI = 262.144 KHz

- #define: **RCC_MSIRANGE_3** **RCC_ICSCR_MSIRANGE_3**

MSI = 524.288 KHz

- #define: **RCC_MSIRANGE_4** **RCC_ICSCR_MSIRANGE_4**

MSI = 1.048 MHz

- #define: **RCC_MSIRANGE_5** **RCC_ICSCR_MSIRANGE_5**

MSI = 2.097 MHz

- #define: **RCC_MSIRANGE_6** **RCC_ICSCR_MSIRANGE_6**

MSI = 4.194 MHz

RCC_MSI_Config

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

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

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)
- #define: **RCC_OSCILLATORTYPE_MSI** ((*uint32_t*)0x00000010)
- #define: **RCC_OSCILLATORTYPE_HSI48** ((*uint32_t*)0x00000020)

RCC_PLLDivider_Factor

- #define: **RCC_PLLDIV_2 RCC_CFGR_PLLDIV2**
- #define: **RCC_PLLDIV_3 RCC_CFGR_PLLDIV3**
- #define: **RCC_PLLDIV_4 RCC_CFGR_PLLDIV4**

RCC_PLLMultiplication_Factor

- #define: **RCC_PLLMUL_3 RCC_CFGR_PLLMUL3**
- #define: **RCC_PLLMUL_4 RCC_CFGR_PLLMUL4**

-
- #define: **RCC_PLLMUL_6 RCC_CFGR_PLLMUL6**
 - #define: **RCC_PLLMUL_8 RCC_CFGR_PLLMUL8**
 - #define: **RCC_PLLMUL_12 RCC_CFGR_PLLMUL12**
 - #define: **RCC_PLLMUL_16 RCC_CFGR_PLLMUL16**
 - #define: **RCC_PLLMUL_24 RCC_CFGR_PLLMUL24**
 - #define: **RCC_PLLMUL_32 RCC_CFGR_PLLMUL32**
 - #define: **RCC_PLLMUL_48 RCC_CFGR_PLLMUL48**

RCC_PLL_Clock_Source

- #define: **RCC_PLLSOURCE_HSI RCC_CFGR_PLLSRC_HSI**
- #define: **RCC_PLLSOURCE_HSE RCC_CFGR_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 RCC_CSR_RTCSEL_LSE***
- #define: ***RCC_RTCCLKSOURCE_LSI RCC_CSR_RTCSEL_LSI***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV2 RCC_CSR_RTCSEL_HSE***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV4 ((uint32_t)RCC_CSR_RTCSEL_HSE | RCC_CR_RTCPRE_0)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV8 ((uint32_t)RCC_CSR_RTCSEL_HSE | RCC_CR_RTCPRE_1)***
- #define: ***RCC_RTCCLKSOURCE_HSE_DIV16 ((uint32_t)RCC_CSR_RTCSEL_HSE | RCC_CR_RTCPRE)***

RCC_System_Clock_Source

- #define: ***RCC_SYSCLKSOURCE_MSI RCC_CFGR_SW_MSI***
- #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)

32 HAL RCC Extension Driver

32.1.1 RCCEEx Firmware driver introduction

32.2 RCCEEx Firmware driver registers structures

32.2.1 RCC_CRSInitTypeDef

RCC_CRSInitTypeDef is defined in the `stm32l0xx_hal_rcc_ex.h`

Data Fields

- `uint32_t Prescaler`
- `uint32_t Source`
- `uint32_t Polarity`
- `uint32_t ReloadValue`
- `uint32_t ErrorLimitValue`
- `uint32_t HSI48CalibrationValue`

Field Documentation

- `uint32_t RCC_CRSInitTypeDef::Prescaler`
 - Specifies the division factor of the SYNC signal. This parameter can be a value of `RCCEEx_CRS_SynchroDivider`
- `uint32_t RCC_CRSInitTypeDef::Source`
 - Specifies the SYNC signal source. This parameter can be a value of `RCCEEx_CRS_SynchroSource`
- `uint32_t RCC_CRSInitTypeDef::Polarity`
 - Specifies the input polarity for the SYNC signal source. This parameter can be a value of `RCCEEx_CRS_SynchroPolarity`
- `uint32_t RCC_CRSInitTypeDef::ReloadValue`
 - Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro
`_HAL_RCC_CRS_CALCULATE_RELOADVALUE(_FTARGET_, _FSYNC_)`
 This parameter must be a number between 0 and 0xFFFF or a value of `RCCEEx_CRS_ReloadValueDefault`
- `uint32_t RCC_CRSInitTypeDef::ErrorLimitValue`
 - Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of `RCCEEx_CRS_ErrorLimitDefault`
- `uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue`
 - Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of `RCCEEx_CRS_HSI48CalibrationDefault`

32.2.2 RCC_CRSSynchroInfoTypeDef

RCC_CRSSynchroInfoTypeDef is defined in the `stm32l0xx_hal_rcc_ex.h`

Data Fields

- *uint32_t ReloadValue*
- *uint32_t HSI48CalibrationValue*
- *uint32_t FreqErrorCapture*
- *uint32_t FreqErrorDirection*

Field Documentation

- *uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue*
 - Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- *uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue*
 - Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- *uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture*
 - Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- *uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection*
 - Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of [RCCEEx_CRS_FreqErrorDirection](#)

32.2.3 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the `stm32l0xx_hal_rcc_ex.h`

Data Fields

- *uint32_t PeriphClockSelection*
- *uint32_t Usart1ClockSelection*
- *uint32_t Usart2ClockSelection*
- *uint32_t Lpuart1ClockSelection*
- *uint32_t I2c1ClockSelection*
- *uint32_t RTCClockSelection*
- *uint32_t UsbClockSelection*
- *uint32_t LptimClockSelection*

Field Documentation

- *uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection*

- The Extended Clock to be configured. This parameter can be a value of [RCCEEx_Periph_Clock_Selection](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection`**
 - USART1 clock source This parameter can be a value of [RCCEEx_USART1_Clock_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection`**
 - USART2 clock source This parameter can be a value of [RCCEEx_USART2_Clock_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection`**
 - LPUART1 clock source This parameter can be a value of [RCCEEx_LPUART_Clock_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection`**
 - I2C1 clock source This parameter can be a value of [RCCEEx_I2C1_Clock_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection`**
 - Specifies RTC Clock Prescalers Selection This parameter can be a value of [RCC_RTC_Clock_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection`**
 - Specifies USB and RNG Clock Selection This parameter can be a value of [RCCEEx_USB_Clock_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::LptimClockSelection`**
 - LPTIM1 clock source This parameter can be a value of [RCCEEx_LPTIM1_Clock_Source](#)

32.2.4 CRS_TypeDef

`CRS_TypeDef` is defined in the `stm32l052xx.h`

Data Fields

- **`__IO uint32_t CR`**
- **`__IO uint32_t CFGR`**
- **`__IO uint32_t ISR`**
- **`__IO uint32_t ICR`**

Field Documentation

- **`__IO uint32_t CRS_TypeDef::CR`**
 - CRS control register, Address offset: 0x00
- **`__IO uint32_t CRS_TypeDef::CFGGR`**
 - CRS configuration register, Address offset: 0x04
- **`__IO uint32_t CRS_TypeDef::ISR`**
 - CRS interrupt and status register, Address offset: 0x08
- **`__IO uint32_t CRS_TypeDef::ICR`**
 - CRS interrupt flag clear register, Address offset: 0x0C

32.3 RCCEx Firmware driver API description

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

32.3.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

- [*HAL_RCCEEx_PeriphCLKConfig\(\)*](#)
- [*HAL_RCCEEx_GetPeriphCLKConfig\(\)*](#)
- [*HAL_RCCEEx_EnableLSECSS\(\)*](#)
- [*HAL_RCCEEx_DisableLSECSS\(\)*](#)
- [*HAL_RCCEEx_CRSConfig\(\)*](#)
- [*HAL_RCCEEx_CRSSoftwareSynchronizationGenerate\(\)*](#)
- [*HAL_RCCEEx_CRSGetSynchronizationInfo\(\)*](#)
- [*HAL_RCCEEx_CRSWaitSynchronization\(\)*](#)

32.3.1.1 HAL_RCCEEx_PeriphCLKConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig (</code> <i>RCC_PeriphCLKInitTypeDef * PeriphClkInit</i>)
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the <code>RCC_PeriphCLKInitTypeDef</code> .
Parameters	<ul style="list-style-type: none"> • PeriphClkInit : pointer to an <code>RCC_PeriphCLKInitTypeDef</code> structure that contains the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, RTC, USB/RNG and LPTIM1 clocks).
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

32.3.1.2 HAL_RCCEEx_GetPeriphCLKConfig

Function Name	<code>void HAL_RCCEEx_GetPeriphCLKConfig (</code> <i>RCC_PeriphCLKInitTypeDef * PeriphClkInit</i>)
Function Description	Get the <code>RCC_ClkInitStruct</code> according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • PeriphClkInit : pointer to an <code>RCC_PeriphCLKInitTypeDef</code> structure that returns the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, RTC, USB/RNG and LPTIM1 clocks).
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

32.3.1.3 HAL_RCCEx_EnableLSECSS

Function Name	void HAL_RCCEx_EnableLSECSS (void)
Function Description	Enables the LSE Clock Security System.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

32.3.1.4 HAL_RCCEx_DisableLSECSS

Function Name	void HAL_RCCEx_DisableLSECSS (void)
Function Description	Disables the LSE Clock Security System.
Parameters	<ul style="list-style-type: none">• None.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

32.3.1.5 HAL_RCCEx_CRSConfig

Function Name	void HAL_RCCEx_CRSConfig (<i>RCC_CRSInitTypeDef</i> * pInit)
Function Description	Start automatic synchronization using polling mode.
Parameters	<ul style="list-style-type: none">• pInit : Pointer on RCC_CRSInitTypeDef structure
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

32.3.1.6 HAL_RCCEEx_CRSSoftwareSynchronizationGenerate

Function Name	void HAL_RCCEEx_CRSSoftwareSynchronizationGenerate (void)
Function Description	Generate the software synchronization event.
Parameters	<ul style="list-style-type: none"> • None.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

32.3.1.7 HAL_RCCEEx_CRSGetSynchronizationInfo

Function Name	void HAL_RCCEEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)
Function Description	Function to return synchronization info.
Parameters	<ul style="list-style-type: none"> • pSynchroInfo : Pointer on RCC_CRSSynchroInfoTypeDef structure
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

32.3.1.8 HAL_RCCEEx_CRSWaitSynchronization

Function Name	RCC_CRSStatusTypeDef HAL_RCCEEx_CRSWaitSynchronization (uint32_t Timeout)
Function Description	This function handles CRS Synchronization Timeout.
Parameters	<ul style="list-style-type: none"> • Timeout : Duration of the timeout
Return values	<ul style="list-style-type: none"> • Combination of Synchronization status This parameter can be a combination of the following values: <ul style="list-style-type: none"> - RCC_CRS_TIMEOUT - RCC_CRS_SYNCOK - RCC_CRS_SYNCWARM - RCC_CRS_SYNCERR - RCC_CRS_SYNCMISS

– ***RCC_CRS_TRIMOV***

Notes

- Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
- If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

32.4 RCCEEx Firmware driver defines

32.4.1 RCCEEx

RCCEEx

RCCEEx_CRS_ErrorLimitDefault

- #define: ***RCC_CRS_ERRORLIMIT_DEFAULT ((uint32_t)0x22)***
Default Frequency error limit

RCCEEx_CRS_Flags

- #define: ***RCC_CRS_FLAG_SYNCOK CRS_ISR_SYNCOKF***
- #define: ***RCC_CRS_FLAG_SYNCWARN CRS_ISR_SYNCWARNF***
- #define: ***RCC_CRS_FLAG_ERR CRS_ISR_ERRF***
- #define: ***RCC_CRS_FLAG_ESYNC CRS_ISR_ESYNCF***
- #define: ***RCC_CRS_FLAG_TRIMOVF CRS_ISR_TRIMOVF***
Trimming overflow or underflow
- #define: ***RCC_CRS_FLAG_SYNCERR CRS_ISR_SYNCERR***
SYNC error
- #define: ***RCC_CRS_FLAG_SYNCMISS CRS_ISR_SYNCMISS***
SYNC missed

RCCEEx_CRS_FreqErrorDirection

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

Upcounting direction, the actual frequency is above the target

- #define: **RCC_CRS_FREQERRORDIR_DOWN** ((*uint32_t*)CRS_ISR_FEDIR)

Downcounting direction, the actual frequency is below the target

RCCEEx_CRS_HSI48CalibrationDefault

- #define: **RCC_CRS_HSI48CALIBRATION_DEFAULT** ((*uint32_t*)0x20)

The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEEx_CRS_Interrupt_Sources

- #define: **RCC_CRS_IT_SYNCOK CRS_ISR_SYNCOKF**

SYNC event OK

- #define: **RCC_CRS_IT_SYNCWARN CRS_ISR_SYNCWARNF**

SYNC warning

- #define: **RCC_CRS_IT_ERR CRS_ISR_ERRF**

error

- #define: **RCC_CRS_IT_ESYNC CRS_ISR_ESYNCF**

Expected SYNC

- #define: **RCC_CRS_IT_TRIMOVF CRS_ISR_TRIMOVF**

Trimming overflow or underflow

- #define: **RCC_CRS_IT_SYNCERR CRS_ISR_SYNCERR**

SYNC error

- #define: **RCC_CRS_IT_SYNCMISS CRS_ISR_SYNCMISS**

SYNC missed

RCCEEx_CRS_ReloadValueDefault

- #define: **RCC_CRS_RELOADVALUE_DEFAULT** ((*uint32_t*)0xBB7F)

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEx_CRS_SynchroDivider

- #define: **RCC_CRS_SYNC_DIV1** ((*uint32_t*)0x00)

Synchro Signal not divided (default)

- #define: **RCC_CRS_SYNC_DIV2 CRS_CFGR_SYNCDIV_0**

Synchro Signal divided by 2

- #define: **RCC_CRS_SYNC_DIV4 CRS_CFGR_SYNCDIV_1**

Synchro Signal divided by 4

- #define: **RCC_CRS_SYNC_DIV8 (CRS_CFGR_SYNCDIV_1 | CRS_CFGR_SYNCDIV_0)**

Synchro Signal divided by 8

- #define: **RCC_CRS_SYNC_DIV16 CRS_CFGR_SYNCDIV_2**

Synchro Signal divided by 16

- #define: **RCC_CRS_SYNC_DIV32 (CRS_CFGR_SYNCDIV_2 | CRS_CFGR_SYNCDIV_0)**

Synchro Signal divided by 32

- #define: **RCC_CRS_SYNC_DIV64 (CRS_CFGR_SYNCDIV_2 | CRS_CFGR_SYNCDIV_1)**

Synchro Signal divided by 64

- #define: **RCC_CRS_SYNC_DIV128 CRS_CFGR_SYNCDIV**

Synchro Signal divided by 128

RCCEx_CRS_SynchroPolarity

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

Synchro Active on rising edge (default)

- #define: **RCC_CRS_SYNC_POLARITY_FALLING CRS_CFGR_SYNCPO**

Synchro Active on falling edge

RCCEx_CRS_SynchroSource

- #define: **RCC_CRS_SYNC_SOURCE_GPIO** ((*uint32_t*)0x00)
Synchro Signal source GPIO
- #define: **RCC_CRS_SYNC_SOURCE_LSE** CRS_CFRG_SYNCSRC_0
Synchro Signal source LSE
- #define: **RCC_CRS_SYNC_SOURCE_USB** CRS_CFRG_SYNCSRC_1
Synchro Signal source USB SOF (default)

RCCEx_Exported_Macros

- #define: **__CRYP_CLK_ENABLE** (RCC->AHBENR |= (RCC_AHBENR_CRYPEN))
After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.
- #define: **__CRYP_CLK_DISABLE** (RCC->AHBENR &= ~ (RCC_AHBENR_CRYPEN))
- #define: **__TSC_CLK_ENABLE** (RCC->AHBENR |= (RCC_AHBENR_TSCEN))
- #define: **__TSC_CLK_DISABLE** (RCC->AHBENR &= ~ (RCC_AHBENR_TSCEN))
- #define: **__RNG_CLK_ENABLE** (RCC->AHBENR |= (RCC_AHBENR_RNGEN))
- #define: **__RNG_CLK_DISABLE** (RCC->AHBENR &= ~ (RCC_AHBENR_RNGEN))
- #define: **__USB_CLK_ENABLE** (RCC->APB1ENR |= (RCC_APB1ENR_USBEN))
After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.
- #define: **__USB_CLK_DISABLE** (RCC->APB1ENR &= ~ (RCC_APB1ENR_USBEN))

- #define: `__CRS_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_CRSEN))`
- #define: `__CRS_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_CRSEN))`
- #define: `__LCD_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_LCDEN))`
- #define: `__LCD_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_LCDEN))`
- #define: `__TIM2_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_TIM2EN))`
- #define: `__TIM6_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_TIM6EN))`
- #define: `__SPI2_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_SPI2EN))`
- #define: `__USART2_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_USART2EN))`
- #define: `__LPUART1_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_LPUART1EN))`
- #define: `__I2C1_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_I2C1EN))`
- #define: `__I2C2_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_I2C2EN))`

- #define: `__DAC_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_DACEN))`
- #define: `__LPTIM1_CLK_ENABLE (RCC->APB1ENR |= (RCC_APB1ENR_LPTIM1EN))`
- #define: `__TIM2_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_TIM2EN))`
- #define: `__TIM6_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_TIM6EN))`
- #define: `__SPI2_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_SPI2EN))`
- #define: `__USART2_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_USART2EN))`
- #define: `__LPUART1_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_LPUART1EN))`
- #define: `__I2C1_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_I2C1EN))`
- #define: `__I2C2_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_I2C2EN))`
- #define: `__DAC_CLK_DISABLE (RCC->APB1ENR &= ~(RCC_APB1ENR_DACEN))`

- `#define: __LPTIM1_CLK_DISABLE (RCC->APB1ENR &= ~ (RCC_APB1ENR_LPTIM1EN))`

- `#define: __TIM21_CLK_ENABLE (RCC->APB2ENR |= (RCC_APB2ENR_TIM21EN))`
After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

- `#define: __TIM22_CLK_ENABLE (RCC->APB2ENR |= (RCC_APB2ENR_TIM22EN))`

- `#define: __FIREWALL_CLK_ENABLE (RCC->APB2ENR |= (RCC_APB2ENR_MIFIEN))`

- `#define: __ADC1_CLK_ENABLE (RCC->APB2ENR |= (RCC_APB2ENR_ADC1EN))`

- `#define: __SPI1_CLK_ENABLE (RCC->APB2ENR |= (RCC_APB2ENR_SPI1EN))`

- `#define: __USART1_CLK_ENABLE (RCC->APB2ENR |= (RCC_APB2ENR_USART1EN))`

- `#define: __TIM21_CLK_DISABLE (RCC->APB2ENR &= ~ (RCC_APB2ENR_TIM21EN))`

- `#define: __TIM22_CLK_DISABLE (RCC->APB2ENR &= ~ (RCC_APB2ENR_TIM22EN))`

- `#define: __FIREWALL_CLK_DISABLE (RCC->APB2ENR &= ~ (RCC_APB2ENR_MIFIEN))`

- #define: **ADC1_CLK_DISABLE** (*RCC->APB2ENR &= ~
(RCC_APB2ENR_ADC1EN)*)
- #define: **SPI1_CLK_DISABLE** (*RCC->APB2ENR &= ~
(RCC_APB2ENR_SPI1EN)*)
- #define: **USART1_CLK_DISABLE** (*RCC->APB2ENR &= ~
(RCC_APB2ENR_USART1EN)*)
- #define: **CRYP_FORCE_RESET** (*RCC->AHBRSTR |=
(RCC_AHBRSTR_CRYPRST)*)
- #define: **CRYP_RELEASE_RESET** (*RCC->AHBRSTR &= ~
(RCC_AHBRSTR_CRYPRST)*)
- #define: **TSC_FORCE_RESET** (*RCC->AHBRSTR |=
(RCC_AHBRSTR_TSCRST)*)
- #define: **TSC_RELEASE_RESET** (*RCC->AHBRSTR &= ~
(RCC_AHBRSTR_TSCRST)*)
- #define: **RNG_FORCE_RESET** (*RCC->AHBRSTR |=
(RCC_AHBRSTR_RNGRST)*)
- #define: **RNG_RELEASE_RESET** (*RCC->AHBRSTR &= ~
(RCC_AHBRSTR_RNGRST)*)
- #define: **TIM2_FORCE_RESET** (*RCC->APB1RSTR |=
(RCC_APB1RSTR_TIM2RST)*)

- #define: **__TIM6_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_TIM6RST))**
- #define: **__LPTIM1_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_LPTIM1RST))**
- #define: **__I2C1_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_I2C1RST))**
- #define: **__I2C2_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_I2C2RST))**
- #define: **__USART2_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_USART2RST))**
- #define: **__LPUART1_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_LPUART1RST))**
- #define: **__SPI2_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_SPI2RST))**
- #define: **__DAC_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_DACRST))**
- #define: **__TIM2_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_TIM2RST))**
- #define: **__TIM6_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_TIM6RST))**

- #define: **__LPTIM1_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_LPTIM1RST))**
- #define: **__I2C1_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_I2C1RST))**
- #define: **__I2C2_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_I2C2RST))**
- #define: **__USART2_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_USART2RST))**
- #define: **__LPUART1_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_LPUART1RST))**
- #define: **__SPI2_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_SPI2RST))**
- #define: **__DAC_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_DACRST))**
- #define: **__USB_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_USBRST))**
- #define: **__USB_RELEASE_RESET (RCC->APB1RSTR &= ~ (RCC_APB1RSTR_USBRST))**
- #define: **__CRS_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_CRSRST))**

- #define: **__CRS_RELEASE_RESET (RCC->APB1RSTR &= ~(RCC_APB1RSTR_CRSRST))**

- #define: **__LCD_FORCE_RESET (RCC->APB1RSTR |= (RCC_APB1RSTR_LCDRST))**

- #define: **__LCD_RELEASE_RESET (RCC->APB1RSTR &= ~(RCC_APB1RSTR_LCDRST))**

- #define: **__USART1_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_USART1RST))**

- #define: **__ADC1_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_ADC1RST))**

- #define: **__SPI1_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_SPI1RST))**

- #define: **__TIM21_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_TIM21RST))**

- #define: **__TIM22_FORCE_RESET (RCC->APB2RSTR |= (RCC_APB2RSTR_TIM22RST))**

- #define: **__USART1_RELEASE_RESET (RCC->APB2RSTR &= ~(RCC_APB2RSTR_USART1RST))**

- #define: **__ADC1_RELEASE_RESET (RCC->APB2RSTR &= ~(RCC_APB2RSTR_ADC1RST))**

- #define: **_SPI1_RELEASE_RESET (RCC->APB2RSTR &= ~ (RCC_APB2RSTR_SPI1RST))**

- #define: **_TIM21_RELEASE_RESET (RCC->APB2RSTR &= ~ (RCC_APB2RSTR_TIM21RST))**

- #define: **_TIM22_RELEASE_RESET (RCC->APB2RSTR &= ~ (RCC_APB2RSTR_TIM22RST))**

- #define: **_TSC_CLK_SLEEP_ENABLE (RCC->AHBSMENR |= (RCC_AHBSMENR_TSCSMEN))**

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

- #define: **_RNG_CLK_SLEEP_ENABLE (RCC->AHBSMENR |= (RCC_AHBSMENR_RNGSMEN))**

- #define: **_TSC_CLK_SLEEP_DISABLE (RCC->AHBSMENR &= ~ (RCC_AHBSMENR_TSCSMEN))**

- #define: **_RNG_CLK_SLEEP_DISABLE (RCC->AHBSMENR &= ~ (RCC_AHBSMENR_RNGSMEN))**

- #define: **_CRYP_CLK_SLEEP_ENABLE (RCC->AHBLPENR |= (RCC_AHBSMENR_CRYPSMEN))**

- #define: **_CRYP_CLK_SLEEP_DISABLE (RCC->AHBLPENR &= ~ (RCC_AHBSMENR_CRYPSMEN))**

- #define: **`__TIM2_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_TIM2SMEN))`**

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

- #define: **`__TIM6_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_TIM6SMEN))`**

- #define: **`__SPI2_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_SPI2SMEN))`**

- #define: **`__USART2_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_USART2SMEN))`**

- #define: **`__LPUART1_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_LPUART1SMEN))`**

- #define: **`__I2C1_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_I2C1SMEN))`**

- #define: **`__I2C2_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_I2C2SMEN))`**

- #define: **`__DAC_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_DACSMEN))`**

- #define: **`__LPTIM1_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_LPTIM1SMEN))`**

- #define: **`__TIM2_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_TIM2SMEN))`**

- #define: **_TIM6_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_TIM6SMEN))**
- #define: **_SPI2_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_SPI2SMEN))**
- #define: **_USART2_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_USART2SMEN))**
- #define: **_LPUART1_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_LPUART1SMEN))**
- #define: **_I2C1_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_I2C1SMEN))**
- #define: **_I2C2_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_I2C2SMEN))**
- #define: **_DAC_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_DACSMEN))**
- #define: **_LPTIM1_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_LPTIM1SMEN))**
- #define: **_USB_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_USBSMEN))**
- #define: **_USB_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_USBSMEN))**

- #define: **`__CRS_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_CRSSMEN))`**
- #define: **`__CRS_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_CRSSMEN))`**
- #define: **`__LCD_CLK_SLEEP_ENABLE (RCC->APB1SMENR |= (RCC_APB1SMENR_LCDSMEN))`**
- #define: **`__LCD_CLK_SLEEP_DISABLE (RCC->APB1SMENR &= ~ (RCC_APB1SMENR_LCDSMEN))`**
- #define: **`__TIM21_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_TIM21SMEN))`**
Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.
- #define: **`__TIM22_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_TIM22SMEN))`**
- #define: **`__ADC1_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_ADC1SMEN))`**
- #define: **`__SPI1_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_SPI1SMEN))`**
- #define: **`__USART1_CLK_SLEEP_ENABLE (RCC->APB2SMENR |= (RCC_APB2SMENR_USART1SMEN))`**

- #define: **`__TIM21_CLK_SLEEP_DISABLE (RCC->APB2SMENR &= ~ (RCC_APB2SMENR_TIM21SMEN))`**

- #define: **`__TIM22_CLK_SLEEP_DISABLE (RCC->APB2SMENR &= ~ (RCC_APB2SMENR_TIM22SMEN))`**

- #define: **`__ADC1_CLK_SLEEP_DISABLE (RCC->APB2SMENR &= ~ (RCC_APB2SMENR_ADC1SMEN))`**

- #define: **`__SPI1_CLK_SLEEP_DISABLE (RCC->APB2SMENR &= ~ (RCC_APB2SMENR_SPI1SMEN))`**

- #define: **`__HAL_RCC_GET_I2C1_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_I2C1SEL)))`**

The clock source can be one of the following values: `RCC_I2C1CLKSOURCE_PCLK1`: PCLK1 selected as I2C1 clock `RCC_I2C1CLKSOURCE_HSI`: HSI selected as I2C1 clock `RCC_I2C1CLKSOURCE_SYSCLK`: System Clock selected as I2C1 clock

- #define: **`__HAL_RCC_GET_USART1_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_USART1SEL)))`**

The clock source can be one of the following values:
`RCC_USART1CLKSOURCE_PCLK2`: PCLK2 selected as USART1 clock
`RCC_USART1CLKSOURCE_HSI`: HSI selected as USART1 clock
`RCC_USART1CLKSOURCE_SYSCLK`: System Clock selected as USART1 clock
`RCC_USART1CLKSOURCE_LSE`: LSE selected as USART1 clock

- #define: **`__HAL_RCC_GET_USART2_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_USART2SEL)))`**

The clock source can be one of the following values:
`RCC_USART2CLKSOURCE_PCLK1`: PCLK1 selected as USART2 clock
`RCC_USART2CLKSOURCE_HSI`: HSI selected as USART2 clock
`RCC_USART2CLKSOURCE_SYSCLK`: System Clock selected as USART2 clock
`RCC_USART2CLKSOURCE_LSE`: LSE selected as USART2 clock

- #define: **`__HAL_RCC_GET_LPUART1_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_LPUART1SEL)))`**

The clock source can be one of the following values:

RCC_LPUART1CLKSOURCE_PCLK1: PCLK1 selected as LPUART1 clock

RCC_LPUART1CLKSOURCE_HSI: HSI selected as LPUART1 clock

RCC_LPUART1CLKSOURCE_SYSCLK: System Clock selected as LPUART1 clock

RCC_LPUART1CLKSOURCE_LSE: LSE selected as LPUART1 clock

- #define: **`__HAL_RCC_GET_LPTIM1_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_LPTIM1SEL)))`**

The clock source can be one of the following values: *RCC_LPTIM1CLKSOURCE_PCLK: PCLK selected as LPUART1 clock*

RCC_LPTIM1CLKSOURCE_LSI : HSI selected as LPUART1 clock

RCC_LPTIM1CLKSOURCE_HSI : System Clock selected as LPUART1 clock

RCC_LPTIM1CLKSOURCE_LSE : LSE selected as LPUART1 clock

- #define: **`__HAL_RCC_GET_USB_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_HSI48SEL)))`**

The clock source can be one of the following values: *RCC_USBCLKSOURCE_HSI48: HSI48 selected as USB clock*

RCC_USBCLKSOURCE_PLLCLK: PLL Clock selected as USB clock

- #define: **`__HAL_RCC_GET_RNG_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_HSI48SEL)))`**

The clock source can be one of the following values: *RCC RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock*

RCC RNGCLKSOURCE_PLLCLK: PLL Clock selected as RNG clock

- #define: **`__HAL_RCC_GET_HSI48M_SOURCE ((uint32_t)(READ_BIT(RCC->CCIPR, RCC_CCIPR_HSI48SEL)))`**

This macro can be replaced by either `__HAL_RCC_GET_RNG_SOURCE` or `__HAL_RCC_GET_USB_SOURCE` to get respectively RNG or UBS clock sources. The clock source can be one of the following values: *RCC_HSI48M_PLL: A dedicated 48MHZ PLL output*

RCC_HSI48M_RC48: 48MHZ issued from internal HSI48 oscillator.

- #define: **`__HAL_RCC_HSISTOP_ENABLE SET_BIT(RCC->CR, RCC_CR_HSIKERON)`**

The Enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE. None

- #define: **`__HAL_RCC_HSISTOP_DISABLE CLEAR_BIT(RCC->CR, RCC_CR_HSIKERON)`**

- #define: **RCC_CRS_IT_ERROR_MASK** ((*uint32_t*)(RCC_CRS_IT_TRIMOVF | RCC_CRS_IT_SYNCERR | RCC_CRS_IT_SYNCMISS))

INTERRUPT : specifies the interrupt pending bit to clear. This parameter can be any combination of the following values: RCC_CRS_IT_SYNCOK RCC_CRS_IT_SYNCWARN RCC_CRS_IT_ERR RCC_CRS_IT_ESYNC RCC_CRS_IT_TRIMOVF RCC_CRS_IT_SYNCERR RCC_CRS_IT_SYNCMISS

- #define: **RCC_CRS_FLAG_ERROR_MASK** ((*uint32_t*)(RCC_CRS_FLAG_TRIMOVF | RCC_CRS_FLAG_SYNCERR | RCC_CRS_FLAG_SYNCMISS))

FLAG : specifies the flag to clear. This parameter can be one of the following values: RCC_CRS_FLAG_SYNCOK RCC_CRS_FLAG_SYNCWARN RCC_CRS_FLAG_ERR RCC_CRS_FLAG_ESYNC RCC_CRS_FLAG_TRIMOVF RCC_CRS_FLAG_SYNCERR RCC_CRS_FLAG_SYNCMISS None

- #define: **__HAL_RCC_CRS_ENABLE_FREQ_ERROR_COUNTER** (CRS->CR |= **CRS_CR_CEN**)

when the CEN bit is set the CRS_CFG register becomes write-protected. None None

- #define: **__HAL_RCC_CRS_DISABLE_FREQ_ERROR_COUNTER** (CRS->CR &= ~**CRS_CR_CEN**)

None None

- #define: **__HAL_RCC_CRS_ENABLE_AUTOMATIC_CALIB** (CRS->CR |= **CRS_CR_AUTOTRIMEN**)

When the AUTOTRIMEN bit is set the CRS_CFG register becomes write-protected. None None

- #define: **__HAL_RCC_CRS_DISABLE_AUTOMATIC_CALIB** (CRS->CR &= ~**CRS_CR_AUTOTRIMEN**)

None None

RCCEx_HSI48M_Clock_Source

- #define: **RCC_HSI48M_PLL** ((*uint32_t*)0x00000000)

- #define: **RCC_HSI48M_RC48 RCC_CCIPR_HSI48SEL**

RCCEx_I2C1_Clock_Source

- #define: **RCC_I2C1CLKSOURCE_PCLK1** ((*uint32_t*)0x00000000)

- #define: **RCC_I2C1CLKSOURCE_SYSCLK RCC_CCIPR_I2C1SEL_0**
- #define: **RCC_I2C1CLKSOURCE_HSI RCC_CCIPR_I2C1SEL_1**

RCCEx_LPTIM1_Clock_Source

- #define: **RCC_LPTIM1CLKSOURCE_PCLK ((uint32_t)0x00000000)**
- #define: **RCC_LPTIM1CLKSOURCE_LSI RCC_CCIPR_LPTIM1SEL_0**
- #define: **RCC_LPTIM1CLKSOURCE_HSI RCC_CCIPR_LPTIM1SEL_1**
- #define: **RCC_LPTIM1CLKSOURCE_LSE RCC_CCIPR_LPTIM1SEL**

RCCEx_LPUART_Clock_Source

- #define: **RCC_LPUART1CLKSOURCE_PCLK1 ((uint32_t)0x00000000)**
- #define: **RCC_LPUART1CLKSOURCE_SYSCLK RCC_CCIPR_LPUART1SEL_0**
- #define: **RCC_LPUART1CLKSOURCE_HSI RCC_CCIPR_LPUART1SEL_1**
- #define: **RCC_LPUART1CLKSOURCE_LSE (RCC_CCIPR_LPUART1SEL_0 | RCC_CCIPR_LPUART1SEL_1)**

RCCEx_LSEDrive_Configuration

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

- #define: **RCC_LSEDRIVE_MEDIUMLOW RCC_CSR_LSEDRV_0**
- #define: **RCC_LSEDRIVE_MEDIUMHIGH RCC_CSR_LSEDRV_1**
- #define: **RCC_LSEDRIVE_HIGH RCC_CSR_LSEDRV**

RCCEx_Periph_Clock_Selection

- #define: **RCC_PERIPHCLK_USART1 ((uint32_t)0x00000001)**
- #define: **RCC_PERIPHCLK_USART2 ((uint32_t)0x00000002)**
- #define: **RCC_PERIPHCLK_LPUART1 ((uint32_t)0x00000004)**
- #define: **RCC_PERIPHCLK_I2C1 ((uint32_t)0x00000008)**
- #define: **RCC_PERIPHCLK_I2C2 ((uint32_t)0x00000010)**
- #define: **RCC_PERIPHCLK_RTC ((uint32_t)0x00000020)**
- #define: **RCC_PERIPHCLK_USB ((uint32_t)0x00000040)**
- #define: **RCC_PERIPHCLK_LPTIM1 ((uint32_t)0x00000080)**

RCCEEx_RNG_Clock_Source

- #define: **RCC RNGCLKSOURCE_HSI48 RCC_CCIPR_HSI48SEL**

- #define: **RCC RNGCLKSOURCE_PLLCLK ((uint32_t)0x00000000)**

RCCEEx_StopWakeUp_Clock

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

- #define: **RCC_StopWakeUpClock_HSI RCC_CFGR_STOPWUCK**

RCCEEx_TIM_PRescaler_Selection

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

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

RCCEEx_USART1_Clock_Source

- #define: **RCC_USART1CLKSOURCE_PCLK2 ((uint32_t)0x00000000)**

- #define: **RCC_USART1CLKSOURCE_SYSCLK RCC_CCIPR_USART1SEL_0**

- #define: **RCC_USART1CLKSOURCE_HSI RCC_CCIPR_USART1SEL_1**

- #define: **RCC_USART1CLKSOURCE_LSE (RCC_CCIPR_USART1SEL_0 | RCC_CCIPR_USART1SEL_1)**

RCCEEx_USART2_Clock_Source

- #define: **RCC_USART2CLKSOURCE_PCLK1 ((uint32_t)0x00000000)**

- #define: **RCC_USART2CLKSOURCE_SYSCLK RCC_CCIPR_USART2SEL_0**
- #define: **RCC_USART2CLKSOURCE_HSI RCC_CCIPR_USART2SEL_1**
- #define: **RCC_USART2CLKSOURCE_LSE (RCC_CCIPR_USART2SEL_0 | RCC_CCIPR_USART2SEL_1)**

RCCEx_USB_Clock_Source

- #define: **RCC_USBCLKSOURCE_HSI48 RCC_CCIPR_HSI48SEL**
- #define: **RCC_USBCLKSOURCE_PLLCLK ((uint32_t)0x00000000)**

33 HAL RNG Generic Driver

33.1.1 RNG Firmware driver introduction

33.2 RNG Firmware driver registers structures

33.2.1 RNG_HandleTypeDef

RNG_HandleTypeDef is defined in the `stm32l0xx_hal_rng.h`

Data Fields

- *RNG_TypeDef * Instance*
– Register base address
- *HAL_LockTypeDef Lock*
– RNG locking object
- *__IO HAL_RNG_StateTypeDef State*
– RNG communication 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

33.2.2 RNG_TypeDef

RNG_TypeDef is defined in the `stm32l052xx.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

33.3 RNG Firmware driver API description

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

33.3.1 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.

33.3.2 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\(\)`](#)

33.3.3 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\(\)`](#)

33.3.4 Peripheral State functions

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

- [`HAL_RNG_GetState\(\)`](#)

33.3.4.1 `HAL_RNG_Init`

Function Name	HAL_StatusTypeDef HAL_RNG_Init (<i>RNG_HandleTypeDef</i> * <i>hrng</i>)
Function Description	Initializes the RNG according to the specified parameters in the <i>RNG_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

33.3.4.2 HAL_RNG_DeInit

Function Name	HAL_StatusTypeDef HAL_RNG_DeInit (<i>RNG_HandleTypeDef</i> * <i>hrng</i>)
Function Description	Deinitializes the RNG peripheral.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

33.3.4.3 HAL_RNG_MspInit

Function Name	void HAL_RNG_MspInit (<i>RNG_HandleTypeDef</i> * <i>hrng</i>)
Function Description	Initializes the RNG MSP.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

33.3.4.4 HAL_RNG_MspDeInit

Function Name	void HAL_RNG_MspDelInit (<i>RNG_HandleTypeDef</i> * hrng)
Function Description	Deinitializes the RNG MSP.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

33.3.4.5 HAL_RNG_GetRandomNumber

Function Name	uint32_t HAL_RNG_GetRandomNumber (<i>RNG_HandleTypeDef</i> * hrng)
Function Description	Returns a 32-bit random number.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • 32-bit random number
Notes	<ul style="list-style-type: none"> • Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

33.3.4.6 HAL_RNG_GetRandomNumber_IT

Function Name	uint32_t HAL_RNG_GetRandomNumber_IT (<i>RNG_HandleTypeDef</i> * hrng)
Function Description	Returns a 32-bit random number with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • 32-bit random number
Notes	<ul style="list-style-type: none"> • None.

33.3.4.7 HAL_RNG_IRQHandler

Function Name	void HAL_RNG_IRQHandler (<i>RNG_HandleTypeDef</i> * hrng)
Function Description	Handles RNG interrupt request.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • 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.

33.3.4.8 HAL_RNG_ReadyCallback

Function Name	void HAL_RNG_ReadyCallback (<i>RNG_HandleTypeDef</i> * hrng)
Function Description	Data Ready callback in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

33.3.4.9 HAL_RNG_ErrorCallback

Function Name	void HAL_RNG_ErrorCallback (<i>RNG_HandleTypeDef</i> * hrng)
---------------	--

Function Description	RNG error callbacks.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

33.3.4.10 HAL_RNG_GetState

Function Name	HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)
Function Description	Returns the RNG state.
Parameters	<ul style="list-style-type: none"> • hrng : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

33.4 RNG Firmware driver defines

33.4.1 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

34 HAL RTC Generic Driver

34.1.1 RTC Firmware driver introduction

34.2 RTC Firmware driver registers structures

34.2.1 RTC_HandleTypeDef

RTC_HandleTypeDef is defined in the `stm32l0xx_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

34.2.2 RTC_InitTypeDef

RTC_InitTypeDef is defined in the `stm32l0xx_hal_rtc.h`

Data Fields

- *uint32_t HourFormat*
- *uint32_t AsynchPrediv*
- *uint32_t SynchPrediv*
- *uint32_t OutPut*
- *uint32_t OutPutRemap*
- *uint32_t OutPutPolarity*
- *uint32_t OutPutType*

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::OutPutRemap***
 - Specifies the remap for RTC output. This parameter can be a value of [*RTC_Output_ALARM_OUT_Remap*](#)
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
 - Specifies the polarity of the output signal. This parameter can be a value of [*RTC_Output_Polarity_Definitions*](#)
- ***uint32_t RTC_InitTypeDef::OutPutType***
 - Specifies the RTC Output Pin mode. This parameter can be a value of [*RTC_Output_Type_ALARM_OUT*](#)

34.2.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the `stm32l0xx_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

34.2.4 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the `stm32l0xx_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*](#)

34.2.5 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the `stm32l0xx_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](#)

34.2.6 RTC_TypeDef

`RTC_TypeDef` is defined in the `stm32l051xx.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`
- `uint32_t RESERVED`
- `__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 TAMPCR`
- `__IO uint32_t ALRMASSR`
- `__IO uint32_t ALRMBSSR`

- `__IO uint32_t OR`
- `__IO uint32_t BKP0R`
- `__IO uint32_t BKP1R`
- `__IO uint32_t BKP2R`
- `__IO uint32_t BKP3R`
- `__IO uint32_t BKP4R`

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
- `uint32_t RTC_TypeDef::RESERVED`
 - Reserved, 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::TAMPCR`
 - RTC tamper 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
- `__IO uint32_t RTC_TypeDef::OR`
 - RTC option register, Address offset 0x4C
- `__IO uint32_t RTC_TypeDef::BKP0R`
 - RTC backup register 0, 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

34.3 RTC Firmware driver API description

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

34.3.1 Backup Domain Operating Condition

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low power modes or under reset).

34.3.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_CSR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

- Software reset, triggered by setting the RTCRST bit in the RCC Control Status register (RCC_CSR).
- Power reset (BOR/POR/PDR).

34.3.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__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.

34.3.4 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.

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 with interrupt mode using the HAL_RTC_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTC_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal, use the HAL_RTC_Init() function.
- RTC_CALIB: this output is 512Hz signal or 1Hz. To enable the RTC_CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- Two pins can be used as RTC_ALARM or RTC_CALIB (PC13, PB14) managed on the RTC_OR register.
- When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured in output alternate function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

TimeStamp configuration

- Enables 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.

Tamper configuration

- Enable the RTC Tamper and Configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTC_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTC_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC_TAMPCCR register.

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.
- The backup registers are reset when a tamper detection event occurs

34.3.5 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.

34.3.6 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\(\)*](#)

34.3.7 RTC Time and Date functions

This section provide functions allowing to control RTC features (Time, Date, Alarm, Timestamp, Tamper, RefClock ...).

- [*HAL_RTC_SetTime\(\)*](#)
- [*HAL_RTC_GetTime\(\)*](#)
- [*HAL_RTC_SetDate\(\)*](#)
- [*HAL_RTC_GetDate\(\)*](#)

34.3.8 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\(\)*](#)

34.3.9 Peripheral Control functions

This subsection provides functions allowing to

- get the RTC state
- poll for alarm, timestamp, tamper or wakeup timer events
- handle alarm, timestamp, tamper or wakeup timer interrupt request.
- [*HAL_RTC_WaitForSynchro\(\)*](#)
- [*RTC_EnterInitMode\(\)*](#)
- [*RTC_BytetoBcd2\(\)*](#)
- [*RTC_Bcd2ToByte\(\)*](#)

34.3.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- [*HAL_RTC_GetState\(\)*](#)

34.3.10.1 HAL_RTC_Init

Function Name	HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

34.3.10.2 HAL_RTC_DeInit

Function Name	HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)
Function Description	Deinitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function doesn't reset the RTC Backup Data registers.

34.3.10.3 HAL_RTC_MspInit

Function Name	void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

34.3.10.4 HAL_RTC_MspDeInit

Function Name	void HAL_RTC_MspDeInit (<i>RTC_HandleTypeDef</i> * hrtc)
Function Description	Deinitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

34.3.10.5 HAL_RTC_SetTime

Function Name	HAL_StatusTypeDef HAL_RTC_SetTime (<i>RTC_HandleTypeDef</i> * hrtc, <i>RTC_TimeTypeDef</i> * sTime, uint32_t Format)
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – FORMAT_BIN : Binary data format – FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

34.3.10.6 HAL_RTC_GetTime

Function Name	HAL_StatusTypeDef HAL_RTC_GetTime (<i>RTC_HandleTypeDef</i> * hrtc, <i>RTC_TimeTypeDef</i> * sTime, uint32_t Format)
Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

	<ul style="list-style-type: none"> • sTime : Pointer to Time structure • Format : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> – FORMAT_BIN : Binary data format – FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers.

34.3.10.7 HAL_RTC_SetDate

Function Name	HAL_StatusTypeDef HAL_RTC_SetDate (<i>RTC_HandleTypeDef</i> * hrtc, <i>RTC_DateTypeDef</i> * sDate, uint32_t Format)
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – FORMAT_BIN : Binary data format – FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

34.3.10.8 HAL_RTC_GetDate

Function Name	HAL_StatusTypeDef HAL_RTC_GetDate (<i>RTC_HandleTypeDef</i> * hrtc, <i>RTC_DateTypeDef</i> * sDate, uint32_t Format)
Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> • 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:

	<ul style="list-style-type: none"> - FORMAT_BIN : Binary data format - FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

34.3.10.9 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	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - FORMAT_BIN : Binary data format - FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

34.3.10.10 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	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - FORMAT_BIN : Binary data format - FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

-
- | | |
|-------|---|
| Notes | <ul style="list-style-type: none"> None. |
|-------|---|

34.3.10.11 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactive the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> – RTC_ALARM_A : AlarmA – RTC_ALARM_B : AlarmB
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

34.3.10.12 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	<ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> – 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: <ul style="list-style-type: none"> – FORMAT_BIN : Binary data format – FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

34.3.10.13 HAL_RTC_AlarmIRQHandler

Function Name	void HAL_RTC_AlarmIRQHandler (<i>RTC_HandleTypeDef</i> * hrtc)
Function Description	This function handles Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

34.3.10.14 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (<i>RTC_HandleTypeDef</i> * hrtc)
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

34.3.10.15 HAL_RTC_PollForAlarmAEvent

Function Name	HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (<i>RTC_HandleTypeDef</i> * hrtc, uint32_t Timeout)
Function Description	This function handles AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

34.3.10.16 HAL_RTC_WaitForSynchro

Function Name	HAL_StatusTypeDef HAL_RTC_WaitForSynchro (<i>RTC_HandleTypeDef * hrtc</i>)
Function Description	Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"> hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> The RTC Resynchronization mode is write protected, use the <code>__HAL_RTC_WRITEPROTECTION_DISABLE()</code> 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.

34.3.10.17 RTC_EnterInitMode

Function Name	HAL_StatusTypeDef RTC_EnterInitMode (<i>RTC_HandleTypeDef * hrtc</i>)
Function Description	Enters the RTC Initialization mode.
Parameters	<ul style="list-style-type: none"> hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> The RTC Initialization mode is write protected, use the <code>__HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.

34.3.10.18 RTC_ByteToBcd2

Function Name	<code>uint8_t RTC_ByteToBcd2 (uint8_t Value)</code>
Function Description	Converts a 2 digit decimal to BCD format.
Parameters	<ul style="list-style-type: none"> • Value : Byte to be converted
Return values	<ul style="list-style-type: none"> • Converted byte
Notes	<ul style="list-style-type: none"> • None.

34.3.10.19 RTC_Bcd2ToByte

Function Name	<code>uint8_t RTC_Bcd2ToByte (uint8_t Value)</code>
Function Description	Converts from 2 digit BCD to Binary.
Parameters	<ul style="list-style-type: none"> • Value : BCD value to be converted
Return values	<ul style="list-style-type: none"> • Converted word
Notes	<ul style="list-style-type: none"> • None.

34.3.10.20 HAL_RTC_GetState

Function Name	<code>HAL_RTCStateTypeDef HAL_RTC_GetState (</code> <i>RTC_HandleTypeDef * hrtc</i> <code>)</code>
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

34.4 RTC Firmware driver defines

34.4.1 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_Flags_Definitions

- #define: **RTC_FLAG_RECALPF** ((*uint32_t*)RTC_ISR_RECALPF)
- #define: **RTC_FLAG_TAMP2F** ((*uint32_t*)RTC_ISR_TAMP2F)

- #define: ***RTC_FLAG_TAMP1F*** ((*uint32_t*)*RTC_ISR_TAMP1F*)
- #define: ***RTC_FLAG_TSOVF*** ((*uint32_t*)*RTC_ISR_TSOVF*)
- #define: ***RTC_FLAG_TSF*** ((*uint32_t*)*RTC_ISR_TSF*)
- #define: ***RTC_FLAG_WUTF*** ((*uint32_t*)*RTC_ISR_WUTF*)
- #define: ***RTC_FLAG_ALRBF*** ((*uint32_t*)*RTC_ISR_ALRBF*)
- #define: ***RTC_FLAG_ALRAF*** ((*uint32_t*)*RTC_ISR_ALRAF*)
- #define: ***RTC_FLAG_INITF*** ((*uint32_t*)*RTC_ISR_INITF*)
- #define: ***RTC_FLAG_RSF*** ((*uint32_t*)*RTC_ISR_RSF*)
- #define: ***RTC_FLAG_INITS*** ((*uint32_t*)*RTC_ISR_INITS*)
- #define: ***RTC_FLAG_SHPF*** ((*uint32_t*)*RTC_ISR_SHPF*)
- #define: ***RTC_FLAG_WUTWF*** ((*uint32_t*)*RTC_ISR_WUTWF*)
- #define: ***RTC_FLAG_ALRBWF*** ((*uint32_t*)*RTC_ISR_ALRBWF*)

- #define: **RTC_FLAG_ALRAWF** ((*uint32_t*)RTC_ISR_ALRAWF)

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*)0x00000000)
- #define: **FORMAT_BCD** ((*uint32_t*)0x00000001)

RTC Interrupts Definitions

- #define: **RTC_IT_TS** ((*uint32_t*)RTC_CR_TSIE)
- #define: **RTC_IT_WUT** ((*uint32_t*)RTC_CR_WUTIE)
- #define: **RTC_IT_ALRA** ((*uint32_t*)RTC_CR_ALRAIE)
- #define: **RTC_IT_ALRB** ((*uint32_t*)RTC_CR_ALRBIE)
- #define: **RTC_IT_TAMP** ((*uint32_t*)RTC_TAMPSCR_TAMPIE)
- #define: **RTC_IT_TAMP1** ((*uint32_t*)RTC_TAMPSCR_TAMP1IE)

- #define: ***RTC_IT_TAMP2 ((uint32_t)RTC_TAMPCR_TAMP2IE)***

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_ALARM_OUT_Remap

- #define: **RTC_OUTPUT_REMAP_PC13** ((uint32_t)0x00000000)
- #define: **RTC_OUTPUT_REMAP_PB14** ((uint32_t)RTC_OR_RTC_OUT_RMP)

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)RTC_CR_OSEL_0)
- #define: **RTC_OUTPUT_ALARMB** ((uint32_t)RTC_CR_OSEL_1)
- #define: **RTC_OUTPUT_WAKEUP** ((uint32_t)RTC_CR_OSEL)

RTC_Output_Type_ALARM_OUT

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

-
- #define: **RTC_OUTPUT_TYPE_PUSHPULL**
((*uint32_t*)RTC_OR_ALARMOUTTYPE)

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)

35 HAL RTC Extension Driver

35.1.1 RTCEx Firmware driver introduction

35.2 RTCEx Firmware driver registers structures

35.2.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the `stm32l0xx_hal_rtc_ex.h`

Data Fields

- *uint32_t Tamper*
- *uint32_t Interrupt*
- *uint32_t Trigger*
- *uint32_t NoErase*
- *uint32_t MaskFlag*
- *uint32_t Filter*
- *uint32_t SamplingFrequency*
- *uint32_t PrechargeDuration*
- *uint32_t TamperPullUp*
- *uint32_tTimeStampOnTamperDetection*

Field Documentation

- *uint32_t RTC_TamperTypeDef::Tamper*
 - Specifies the Tamper Pin. This parameter can be a value of [*RTCEEx_Tamper_Pins_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::Interrupt*
 - Specifies the Tamper Interrupt. This parameter can be a value of [*RTCEEx_Tamper_Interrupt_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::Trigger*
 - Specifies the Tamper Trigger. This parameter can be a value of [*RTCEEx_Tamper_Trigger_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::NoErase*
 - Specifies the Tamper no erase mode. This parameter can be a value of [*RTCEEx_Tamper_EraseBackUp_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::MaskFlag*
 - Specifies the Tamper Flag masking. This parameter can be a value of [*RTCEEx_Tamper_MaskFlag_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::Filter*
 - Specifies the RTC Filter Tamper. This parameter can be a value of [*RTCEEx_Tamper_Filter_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::SamplingFrequency*
 - Specifies the sampling frequency. This parameter can be a value of [*RTCEEx_Tamper_Sampling_Frequencies_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::PrechargeDuration*

- Specifies the Precharge Duration . This parameter can be a value of [*RTCEEx_Tamper_Pin_Precharge_Duration_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::TamperPullUp*
 - Specifies the Tamper PullUp . This parameter can be a value of [*RTCEEx_Tamper_PullUP_Definitions*](#)
- *uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection*
 - Specifies the TimeStampOnTamperDetection. This parameter can be a value of [*RTCEEx_Tamper_TimeStampOnTamperDetection_Definitions*](#)

35.3 RTCEEx Firmware driver API description

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

35.3.1 RTC TimeStamp and Tamper functions

- [*HAL_RTCEEx_SetTimeStamp\(\)*](#)
- [*HAL_RTCEEx_SetTimeStamp_IT\(\)*](#)
- [*HAL_RTCEEx_DeactivateTimeStamp\(\)*](#)
- [*HAL_RTCEEx_GetTimeStamp\(\)*](#)
- [*HAL_RTCEEx_SetTamper\(\)*](#)
- [*HAL_RTCEEx_SetTamper_IT\(\)*](#)
- [*HAL_RTCEEx_DeactivateTamper\(\)*](#)
- [*HAL_RTCEEx_SetWakeUpTimer\(\)*](#)
- [*HAL_RTCEEx_SetWakeUpTimer_IT\(\)*](#)
- [*HAL_RTCEEx_DeactivateWakeUpTimer\(\)*](#)
- [*HAL_RTCEEx_GetWakeUpTimer\(\)*](#)
- [*HAL_RTCEEx_BKUPWrite\(\)*](#)
- [*HAL_RTCEEx_BKUPRead\(\)*](#)
- [*HAL_RTCEEx_SetSmoothCalib\(\)*](#)
- [*HAL_RTCEEx_SetSynchroShift\(\)*](#)
- [*HAL_RTCEEx_SetCalibrationOutPut\(\)*](#)
- [*HAL_RTCEEx_DeactivateCalibrationOutPut\(\)*](#)
- [*HAL_RTCEEx_SetRefClock\(\)*](#)
- [*HAL_RTCEEx_DeactivateRefClock\(\)*](#)
- [*HAL_RTCEEx_EnableBypassShadow\(\)*](#)
- [*HAL_RTCEEx_DisableBypassShadow\(\)*](#)
- [*HAL_RTCEEx_TamperTimeStampIRQHandler\(\)*](#)
- [*HAL_RTCEEx_WakeUpTimerIRQHandler\(\)*](#)
- [*HAL_RTCEEx_AlarmBEventCallback\(\)*](#)
- [*HAL_RTCEEx_TimeStampEventCallback\(\)*](#)
- [*HAL_RTCEEx_Tamper1EventCallback\(\)*](#)
- [*HAL_RTCEEx_Tamper2EventCallback\(\)*](#)
- [*HAL_RTCEEx_WakeUpTimerEventCallback\(\)*](#)
- [*HAL_RTCEEx_PollForAlarmBEvent\(\)*](#)
- [*HAL_RTCEEx_PollForTimeStampEvent\(\)*](#)
- [*HAL_RTCEEx_PollForTamper1Event\(\)*](#)
- [*HAL_RTCEEx_PollForTamper2Event\(\)*](#)
- [*HAL_RTCEEx_PollForWakeUpTimerEvent\(\)*](#)

35.3.1.1 HAL_RTCEx_SetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge,</code> <code>uint32_t RTC_TimeStampPin)</code>
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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: <ul style="list-style-type: none"> – RTC_TIMESTAMPPIN_PC13 : PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is used.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This API must be called before enabling the TimeStamp feature.

35.3.1.2 HAL_RTCEx_SetTimeStamp_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge,</code> <code>uint32_t RTC_TimeStampPin)</code>
Function Description	Sets TimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Parameters	<ul style="list-style-type: none"> • TimeStampEdge : Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: <ul style="list-style-type: none"> – 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. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is used.

Return values

- **HAL status**

Notes

- This API must be called before enabling theTimeStamp feature.

35.3.1.3 HAL_RTCEx_DeactivateTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (<i>RTC_HandleTypeDef * hrtc</i>)
Function Description	Deactivates TimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.4 HAL_RTCEx_GetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (<i>RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format</i>)
Function Description	Gets the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.5 HAL_RTCEx_SetTamper

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTamper (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</code>
Function Description	Sets Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper : Pointer to Tamper Structure.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we disable the tamper interrupt for all tampers.

35.3.1.6 HAL_RTCEx_SetTamper_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</code>
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper : Pointer to RTC Tamper.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we force the tamper interrupt for all tampers.

35.3.1.7 HAL_RTCEx_DeactivateTamper

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t Tamper)</code>
---------------	--

Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.8 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	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • WakeUpCounter : Wake up counter • WakeUpClock : Wake up clock
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.9 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	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • WakeUpCounter : Wake up counter • WakeUpClock : Wake up clock
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.10 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_DeactivateWakeUpTimer (</code> <i>RTC_HandleTypeDef</i> * <code>hrtc)</code>
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.11 HAL_RTCEx_GetWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_GetWakeUpTimer (</code> <i>RTC_HandleTypeDef</i> * <code>hrtc)</code>
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • Counter value
Notes	<ul style="list-style-type: none"> • None.

35.3.1.12 HAL_RTCEx_BKUPWrite

Function Name	<code>void HAL_RTCEx_BKUPWrite (</code> <i>RTC_HandleTypeDef</i> * <code>hrtc,</code> <code>uint32_t BackupRegister, uint32_t Data)</code>
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> • 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.

	<ul style="list-style-type: none"> • Data : Data to be written in the specified RTC Backup data register.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.13 HAL_RTCEx_BKUPRead

Function Name	<code>uint32_t HAL_RTCEx_BKUPRead (<i>RTC_HandleTypeDef</i> * hrtc, uint32_t BackupRegister)</code>
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a <i>RTC_HandleTypeDef</i> 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	<ul style="list-style-type: none"> • Read value
Notes	<ul style="list-style-type: none"> • None.

35.3.1.14 HAL_RTCEx_SetSmoothCalib

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (<i>RTC_HandleTypeDef</i> * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)</code>
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a <i>RTC_HandleTypeDef</i> 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 : <ul style="list-style-type: none"> – <i>RTC_SMOOTHCALIB_PERIOD_32SEC</i> : The smooth calibration periode is 32s. – <i>RTC_SMOOTHCALIB_PERIOD_16SEC</i> : The smooth calibration periode is 16s. – <i>RTC_SMOOTHCALIB_PERIOD_8SEC</i> : The smooth calibartion periode is 8s. • SmoothCalibPlusPulses : Select to Set or reset the CALP

	<p>bit. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> - <i>RTC_SMOOTHCALIB_PLUSPULSES_SET</i>: Add one RTCCLK puls every 2^{11} pulses. - <i>RTC_SMOOTHCALIB_PLUSPULSES_RESET</i>: No RTCCLK pulses are added. <ul style="list-style-type: none"> • SmoothCalibMinusPulsesValue : Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue mut be equal to 0.

35.3.1.15 HAL_RTCEx_SetSynchroShift

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (</code> <i>RTC_HandleTypeDef</i> * hrtc, <code>uint32_t ShiftAdd1S, uint32_t ShiftSubFS)</code>
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> • 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 : <ul style="list-style-type: none"> - <i>RTC_SHIFTADD1S_SET</i> : Add one second to the clock calendar. - <i>RTC_SHIFTADD1S_RESET</i> : No effect. • ShiftSubFS : Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When REFCKON is set, firmware must not write to Shift control register.

35.3.1.16 HAL_RTCEx_SetCalibrationOutPut

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (</code>
---------------	---

RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - RTC_CALIBOUTPUT_512HZ : A signal has a regular waveform at 512Hz. - RTC_CALIBOUTPUT_1HZ : A signal has a regular waveform at 1Hz.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.17 HAL_RTCEx_DeactivateCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (<i>RTC_HandleTypeDef * hrtc</i>)
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.18 HAL_RTCEx_SetRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetRefClock (<i>RTC_HandleTypeDef * hrtc</i>)
Function Description	Enables the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

35.3.1.19 HAL_RTCEx_DeactivateRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

35.3.1.20 HAL_RTCEx_EnableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Enables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

35.3.1.21 HAL_RTCEx_DisableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Disables the Bypass Shadow feature.

Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

35.3.1.22 HAL_RTCEx_TamperTimeStampIRQHandler

Function Name	void HAL_RTCEx_TamperTimeStampIRQHandler (<i>RTC_HandleTypeDef * hrtc)</i>
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.23 HAL_RTCEx_WakeUpTimerIRQHandler

Function Name	void HAL_RTCEx_WakeUpTimerIRQHandler (<i>RTC_HandleTypeDef * hrtc)</i>
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.24 HAL_RTCEx_AlarmBEventCallback

Function Name	void HAL_RTCEx_AlarmBEventCallback (
---------------	---

RTC_HandleTypeDef * hrtc)

Function Description	Alarm B callback.
Parameters	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.25 HAL_RTCEx_TimeStampEventCallback

Function Name	void HAL_RTCEx_TimeStampEventCallback (<i>RTC_HandleTypeDef * hrtc)</i>
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.26 HAL_RTCEx_Tamper1EventCallback

Function Name	void HAL_RTCEx_Tamper1EventCallback (<i>RTC_HandleTypeDef * hrtc)</i>
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.27 HAL_RTCEx_Tamper2EventCallback

Function Name	void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.28 HAL_RTCEx_WakeUpTimerEventCallback

Function Name	void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

35.3.1.29 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	<ul style="list-style-type: none"> • hrtc : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.30 HAL_RTCEx_PollForTimeStampEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (<i>RTC_HandleTypeDef</i> * hrtc, uint32_t Timeout)
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.31 HAL_RTCEx_PollForTamper1Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (<i>RTC_HandleTypeDef</i> * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.32 HAL_RTCEx_PollForTamper2Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (<i>RTC_HandleTypeDef</i> * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper2 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.3.1.33 HAL_RTCEx_PollForWakeUpTimerEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent(<i>RTC_HandleTypeDef</i> * hrtc, uint32_t Timeout)
Function Description	This function handles Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none"> • hrtc : RTC handle • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

35.4 RTCEx Firmware driver defines

35.4.1 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)***

RTCEEx_Calib_Output_selection_Definitions

- #define: ***RTC_CALIBOUTPUT_512HZ ((uint32_t)0x00000000)***
- #define: ***RTC_CALIBOUTPUT_1HZ ((uint32_t)0x00080000)***

RTCEEx_Digital_Calibration_Definitions

- #define: ***RTC_CALIBSIGN_POSITIVE ((uint32_t)0x00000000)***
- #define: ***RTC_CALIBSIGN_NEGATIVE ((uint32_t)0x00000080)***

RTCEEx_Smooth_calib_period_Definitions

- #define: ***RTC_SMOOTHCALIB_PERIOD_32SEC ((uint32_t)0x00000000)***
If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else $2^{\text{exp}20}$ RTCCLK seconds
- #define: ***RTC_SMOOTHCALIB_PERIOD_16SEC ((uint32_t)0x00002000)***
If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else $2^{\text{exp}19}$ RTCCLK seconds
- #define: ***RTC_SMOOTHCALIB_PERIOD_8SEC ((uint32_t)0x00004000)***
If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else $2^{\text{exp}18}$ RTCCLK seconds

RTCEEx_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 substituted during a 32-second window = CALM[8:0]

RTCEEx_Tamper_EraseBackUp_Definitions

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

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

RTCEEx_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 level.

RTCEEx_Tamper_Interrupt_Definitions

- #define: ***RTC_TAMPER1_INTERRUPT RTC_TAMPSCR_TAMP1IE***

- #define: ***RTC_TAMPER2_INTERRUPT RTC_TAMPSCR_TAMP2IE***

RTCEEx_Tamper_MaskFlag_Definitions

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

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

RTCEEx_Tamper_Pins_Definitions

- #define: ***RTC_TAMPER_1 RTC_TAMPSCR_TAMP1E***

- #define: ***RTC_TAMPER_2 RTC_TAMPSCR_TAMP2E***

RTCEEx_Tamper_Pin_Preload_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

RTCEEx_Tamper_PullUP_Definitions

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

TimeStamp on Tamper Detection event saved

- #define: ***RTC_TAMPER_PULLUP_DISABLE ((uint32_t)RTC_TAMPSCR_TAMPPUDIS)***

TimeStamp on Tamper Detection event is not saved

RTCEEx_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

RTCE_X_Tamper_TimeStampOnTamperDetection_Definitions

- #define: ***RTC_TIMESTAMPONTAMPERDETECTION_ENABLE***
((*uint32_t*)RTC_TAMPSCR_TAMPTS)

TimeStamp on Tamper Detection event saved

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

TimeStamp on Tamper Detection event is not saved

RTCE_X_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_HIGHLVEL***
RTC_TAMPERTRIGGER_FALLINGEDGE

RTCEx_TimeStamp_Pin_Selection

- #define: ***RTC_TIMESTAMPPIN_PC13 ((uint32_t)0x00000000)***

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)

36 HAL SMARTCARD Generic Driver

36.1 SMARTCARD Firmware driver introduction

36.2 SMARTCARD Firmware driver registers structures

36.2.1 SMARTCARD_HandleTypeDef

SMARTCARD_HandleTypeDef is defined in the `stm32l0xx_hal_smartcard.h`

Data Fields

- `USART_TypeDef * Instance`
- `SMARTCARD_InitTypeDef Init`
- `SMARTCARD_AdvFeatureInitTypeDef AdvancedInit`
- `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`
- `SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit`
- `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`

36.2.2 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the `stm32l0xx_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 OneBitSampling`
- `uint32_t Prescaler`
- `uint32_t GuardTime`
- `uint32_t NACKState`
- `uint32_t TimeOutEnable`
- `uint32_t TimeOutValue`
- `uint32_t BlockLength`
- `uint32_t AutoRetryCount`

Field Documentation

- **`uint32_t SMARTCARD_InitTypeDef::BaudRate`**
 - Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hsc->Init.BaudRate)))
- **`uint32_t SMARTCARD_InitTypeDef::WordLength`**
 - Specifies the number of data bits transmitted or received in a frame. This parameter SMARTCARD_Word_Length can only be set to 9 (8 data + 1 parity bits).
- **`uint32_t SMARTCARD_InitTypeDef::StopBits`**
 - Specifies the number of stop bits SMARTCARD_Stop_Bits. Only 1.5 stop bits are authorized in SmartCard mode.
- **`uint32_t SMARTCARD_InitTypeDef::Parity`**
 - Specifies the parity mode. This parameter can be a value of **SMARTCARD_Parity**
- **`uint32_t SMARTCARD_InitTypeDef::Mode`**
 - Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of **SMARTCARD_Mode**
- **`uint32_t SMARTCARD_InitTypeDef::CLKPolarity`**
 - Specifies the steady state of the serial clock. This parameter can be a value of **SMARTCARD_Clock_Polarity**
- **`uint32_t SMARTCARD_InitTypeDef::CLKPhase`**
 - Specifies the clock transition on which the bit capture is made. This parameter can be a value of **SMARTCARD_Clock_Phase**
- **`uint32_t SMARTCARD_InitTypeDef::CLKLastBit`**

- Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of **SMARTCARD_Last_Bit**
- **uint32_t SMARTCARD_InitTypeDef::OneBitSampling**
 - Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of **SMARTCARD_OneBit_Sampling**
- **uint32_t SMARTCARD_InitTypeDef::Prescaler**
 - Specifies the SmartCard Prescaler
- **uint32_t SMARTCARD_InitTypeDef::GuardTime**
 - Specifies the SmartCard Guard Time
- **uint32_t SMARTCARD_InitTypeDef::NACKState**
 - Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of **SMARTCARD_NACK_Enable**
- **uint32_t SMARTCARD_InitTypeDef::TimeOutEnable**
 - Specifies whether the receiver timeout is enabled. This parameter can be a value of **SMARTCARD_Timeout_Enable**
- **uint32_t SMARTCARD_InitTypeDef::TimeOutValue**
 - Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- **uint32_t SMARTCARD_InitTypeDef::BlockLength**
 - Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- **uint32_t SMARTCARD_InitTypeDef::AutoRetryCount**
 - Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

36.2.3 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the `stm32l0xx_hal_smartcard.h`

Data Fields

- **uint32_t AdvFeatureInit**
- **uint32_t TxPinLevelInvert**
- **uint32_t RxPinLevelInvert**
- **uint32_t DataInvert**
- **uint32_t Swap**
- **uint32_t OverrunDisable**
- **uint32_t DMADisableonRxError**
- **uint32_t MSBFirst**

Field Documentation

- **uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit**
 - Specifies which advanced SMARTCARD features are initialized. Several advanced features may be initialized at the same time. This parameter can be a value of **SMARTCARD_Advanced_Features_Initialization_Type**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert`
 - Specifies whether the TX pin active level is inverted. This parameter can be a value of `SMARTCARD_Tx_Inv`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert`
 - Specifies whether the RX pin active level is inverted. This parameter can be a value of `SMARTCARD_Rx_Inv`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert`
 - Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of `SMARTCARD_Data_Inv`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap`
 - Specifies whether TX and RX pins are swapped. This parameter can be a value of `SMARTCARD_Rx_Tx_Swap`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable`
 - Specifies whether the reception overrun detection is disabled. This parameter can be a value of `SMARTCARD_Overrun_Disable`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError`
 - Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of `SMARTCARD_DMA_Disable_on_Rx_Error`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst`
 - Specifies whether MSB is sent first on UART line. This parameter can be a value of `SMARTCARD_MSB_First`

36.2.4 USART_TypeDef

`USART_TypeDef` is defined in the `stm32l051xx.h`

Data Fields

- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t CR3`
- `__IO uint32_t BRR`
- `__IO uint16_t GTPR`
- `uint16_t RESERVED2`
- `__IO uint32_t RTOR`
- `__IO uint16_t RQR`
- `uint16_t RESERVED3`
- `__IO uint32_t ISR`
- `__IO uint32_t ICR`
- `__IO uint16_t RDR`
- `uint16_t RESERVED4`
- `__IO uint16_t TDR`
- `uint16_t RESERVED5`

Field Documentation

- `__IO uint32_t USART_TypeDef::CR1`
 - USART Control register 1, Address offset: 0x00
- `__IO uint32_t USART_TypeDef::CR2`
 - USART Control register 2, Address offset: 0x04

- **`__IO uint32_t USART_TypeDef::CR3`**
 - USART Control register 3, Address offset: 0x08
- **`__IO uint32_t USART_TypeDef::BRR`**
 - USART Baud rate register, Address offset: 0x0C
- **`__IO uint16_t USART_TypeDef::GTPR`**
 - USART Guard time and prescaler register, Address offset: 0x10
- **`uint16_t USART_TypeDef::RESERVED2`**
 - Reserved, 0x12
- **`__IO uint32_t USART_TypeDef::RTOR`**
 - USART Receiver Time Out register, Address offset: 0x14
- **`__IO uint16_t USART_TypeDef::RQR`**
 - USART Request register, Address offset: 0x18
- **`uint16_t USART_TypeDef::RESERVED3`**
 - Reserved, 0x1A
- **`__IO uint32_t USART_TypeDef::ISR`**
 - USART Interrupt and status register, Address offset: 0x1C
- **`__IO uint32_t USART_TypeDef::ICR`**
 - USART Interrupt flag Clear register, Address offset: 0x20
- **`__IO uint16_t USART_TypeDef::RDR`**
 - USART Receive Data register, Address offset: 0x24
- **`uint16_t USART_TypeDef::RESERVED4`**
 - Reserved, 0x26
- **`__IO uint16_t USART_TypeDef::TDR`**
 - USART Transmit Data register, Address offset: 0x28
- **`uint16_t USART_TypeDef::RESERVED5`**
 - Reserved, 0x2A

36.3 SMARTCARD Firmware driver API description

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

36.3.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
 - Baud Rate
 - Parity: parity should be enabled, Frame Length is fixed to 8 bits plus parity.
 - Receiver/transmitter modes
 - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
 - Prescaler value
 - Guard bit time
 - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
 - TX and/or RX pin level inversion
 - data logical level inversion
 - RX and TX pins swap
 - RX overrun detection disabling
 - DMA disabling on RX error
 - MSB first on communication line

- Time out enabling (and if activated, timeout value)
- Block length
- Auto-retry counter

The HAL_SMARTCARD_Init() API follow respectively the USART (a)synchronous configuration procedures (details for the procedures are available in reference manual).

- [`HAL_SMARTCARD_Init\(\)`](#)
- [`HAL_SMARTCARD_DelInit\(\)`](#)
- [`HAL_SMARTCARD_MspInit\(\)`](#)
- [`HAL_SMARTCARD_MspDelInit\(\)`](#)

36.3.2 IO operation functions

- [`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\(\)`](#)

36.3.3 Peripheral State functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARD_GetState() API is helpful to check in run-time the state of the SMARTCARD peripheral
- SMARTCARD_SetConfig() API configures the SMARTCARD peripheral
- SMARTCARD_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- [`HAL_SMARTCARD_GetState\(\)`](#)
- [`HAL_SMARTCARD_GetError\(\)`](#)

36.3.3.1 HAL_SMARTCARD_Init

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsc)</code>
Function Description	Initializes the SMARTCARD mode according to the specified parameters in the SMARTCARD_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

36.3.3.2 HAL_SMARTCARD_DelInit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_DelInit (SMARTCARD_HandleTypeDef * hsc)
Function Description	Deinitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none">• hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

36.3.3.3 HAL_SMARTCARD_MspInit

Function Name	void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none">• hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

36.3.3.4 HAL_SMARTCARD_MspDelInit

Function Name	void HAL_SMARTCARD_MspDelInit (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD MSP Delinit.
Parameters	<ul style="list-style-type: none">• hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

36.3.3.5 HAL_SMARTCARD_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Transmit (</code> <code>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • pData : pointer to data buffer • Size : amount of data to be sent • Timeout : : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

36.3.3.6 HAL_SMARTCARD_Receive

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Receive (</code> <code>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • pData : pointer to data buffer • Size : amount of data to be received • Timeout : : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

36.3.3.7 HAL_SMARTCARD_Transmit_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (</code> <code>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</code>
---------------	---

	Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • pData : pointer to data buffer • Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

36.3.3.8 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 interrupt mode.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • pData : pointer to data buffer • Size : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

36.3.3.9 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 DMA mode.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • pData : pointer to data buffer • Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

36.3.3.10 HAL_SMARTCARD_Receive_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • pData : pointer to data buffer • Size : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

36.3.3.11 HAL_SMARTCARD_IRQHandler

Function Name	<code>void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)</code>
Function Description	SMARTCARD interrupt requests handling.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

36.3.3.12 HAL_SMARTCARD_TxCpltCallback

Function Name	<code>void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)</code>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None.

Notes	<ul style="list-style-type: none">None.
-------	---

36.3.3.13 HAL_SMARTCARD_RxCpltCallback

Function Name	void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDefDef * hsc)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

36.3.3.14 HAL_SMARTCARD_ErrorCallback

Function Name	void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDefDef * hsc)
Function Description	SMARTCARD error callbacks.
Parameters	<ul style="list-style-type: none">hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none">None.
Notes	<ul style="list-style-type: none">None.

36.3.3.15 HAL_SMARTCARD_GetState

Function Name	HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDefDef * hsc)
Function Description	return the SMARTCARD state
Parameters	<ul style="list-style-type: none">hsc : SMARTCARD handle

Return values	<ul style="list-style-type: none"> HAL state
Notes	<ul style="list-style-type: none"> None.

36.3.3.16 HAL_SMARTCARD_GetError

Function Name	<code>uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsc)</code>
Function Description	Return the SMARTCARD error code.
Parameters	<ul style="list-style-type: none"> hsc : : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.
Return values	<ul style="list-style-type: none"> SMARTCARD Error Code
Notes	<ul style="list-style-type: none"> None.

36.4 SMARTCARD Firmware driver defines

36.4.1 SMARTCARD

SMARTCARD

SMARTCARD_Advanced_Features_Initialization_Type

- #define: `SMARTCARD_ADVFEATURE_NO_INIT ((uint32_t)0x00000000)`
- #define: `SMARTCARD_ADVFEATURE_TXINVERT_INIT ((uint32_t)0x00000001)`
- #define: `SMARTCARD_ADVFEATURE_RXINVERT_INIT ((uint32_t)0x00000002)`
- #define: `SMARTCARD_ADVFEATURE_DATAINVERT_INIT
((uint32_t)0x00000004)`

- #define: **SMARTCARD_ADVFEATURE_SWAP_INIT** ((*uint32_t*)0x00000008)
- #define: **SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT** ((*uint32_t*)0x00000010)
- #define: **SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT** ((*uint32_t*)0x00000020)
- #define: **SMARTCARD_ADVFEATURE_MSBFIRST_INIT** ((*uint32_t*)0x00000080)

SMARTCARD_Clock_Phase

- #define: **SMARTCARD_PHASE_1EDGE** ((*uint32_t*)0x0000)
- #define: **SMARTCARD_PHASE_2EDGE** ((*uint32_t*)**USART_CR2_CPHA**)

SMARTCARD_Clock_Polarity

- #define: **SMARTCARD_POLARITY_LOW** ((*uint32_t*)0x0000)
- #define: **SMARTCARD_POLARITY_HIGH** ((*uint32_t*)**USART_CR2_CPOL**)

SMARTCARD_CR3_SCAR_CNT_LSB_POS

- #define: **SMARTCARD_CR3_SCARCNT_LSB_POS** ((*uint32_t*) 17)

SMARTCARD_Data_Inv

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

-
- #define: **SMARTCARD_ADVFEATURE_DATAINV_ENABLE** ((*uint32_t*)USART_CR2_DATAINV)

SMARTCARD_DMA_Disable_on_Rx_Error

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

- #define: **SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR** ((*uint32_t*)USART_CR3_DDRE)

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_RXACK** ((*uint32_t*)0x00400000)
- #define: **SMARTCARD_FLAG_TEACK** ((*uint32_t*)0x00200000)
- #define: **SMARTCARD_FLAG_BUSY** ((*uint32_t*)0x00010000)
- #define: **SMARTCARD_FLAG_EOBF** ((*uint32_t*)0x00001000)
- #define: **SMARTCARD_FLAG_RTOF** ((*uint32_t*)0x00000800)
- #define: **SMARTCARD_FLAG_TXE** ((*uint32_t*)0x00000080)

- #define: **SMARTCARD_FLAG_TC** ((*uint32_t*)0x00000040)
- #define: **SMARTCARD_FLAG_RXNE** ((*uint32_t*)0x00000020)
- #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_GTPR_GT_LSBPOS

- #define: **SMARTCARD_GTPR_GT_LSB_POS** ((*uint32_t*) 8)

SMARTCARD_Interruption_Mask

- #define: **SMARTCARD_IT_MASK** ((*uint16_t*)0x001F)

SMARTCARD Interrupt definition

- #define: **SMARTCARD_IT_PE** ((*uint16_t*)0x0028)
- #define: **SMARTCARD_IT_TXE** ((*uint16_t*)0x0727)
- #define: **SMARTCARD_IT_TC** ((*uint16_t*)0x0626)

- #define: **SMARTCARD_IT_RXNE** ((*uint16_t*)0x0525)
- #define: **SMARTCARD_IT_ERR** ((*uint16_t*)0x0060)
- #define: **SMARTCARD_IT_ORE** ((*uint16_t*)0x0300)
- #define: **SMARTCARD_IT_NE** ((*uint16_t*)0x0200)
- #define: **SMARTCARD_IT_FE** ((*uint16_t*)0x0100)
- #define: **SMARTCARD_IT_EOB** ((*uint16_t*)0x0C3B)
- #define: **SMARTCARD_IT_RTO** ((*uint16_t*)0x0B3A)

SMARTCARD_IT_CLEAR_Flags

- #define: **SMARTCARD_CLEAR_PEF USART_ICR_PECF**
Parity Error Clear Flag
- #define: **SMARTCARD_CLEAR_FEF USART_ICR_FECF**
Framing Error Clear Flag
- #define: **SMARTCARD_CLEAR_NEF USART_ICR_NCF**
Noise detected Clear Flag
- #define: **SMARTCARD_CLEAR_OREF USART_ICR_ORECF**
OverRun Error Clear Flag

- #define: **SMARTCARD_CLEAR_TCF USART_ICR_TCCF**

Transmission Complete Clear Flag

- #define: **SMARTCARD_CLEAR_RTOF USART_ICR_RTOCF**

Receiver Time Out Clear Flag

- #define: **SMARTCARD_CLEAR_EOBF USART_ICR_EOBCF**

End Of Block Clear Flag

SMARTCARD_Last_Bit

- #define: **SMARTCARD_LASTBIT_DISABLE ((uint32_t)0x0000)**

- #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_MSB_First

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

- #define: **SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE ((uint32_t)USART_CR2_MSBFIRST)**

SMARTCARD_NACK_Enable

- #define: **SMARTCARD_NACK_ENABLED** ((*uint32_t*)USART_CR3_NACK)

- #define: **SMARTCARD_NACK_DISABLED** ((*uint32_t*)0x0000)

SMARTCARD_OneBit_Sampling

- #define: **SMARTCARD_ONEBIT_SAMPLING_DISABLED** ((*uint32_t*)0x0000)
- #define: **SMARTCARD_ONEBIT_SAMPLING_ENABLED** ((*uint32_t*)USART_CR3_ONEBIT)

SMARTCARD_Overrun_Disable

- #define: **SMARTCARD_ADVFEATURE_OVERRUN_ENABLE** ((*uint32_t*)0x00000000)
- #define: **SMARTCARD_ADVFEATURE_OVERRUN_DISABLE** ((*uint32_t*)USART_CR3_OVRDIS)

SMARTCARD_Parity

- #define: **SMARTCARD_PARITY_EVEN** ((*uint32_t*)USART_CR1_PCE)
- #define: **SMARTCARD_PARITY_ODD** ((*uint32_t*)(USART_CR1_PCE | USART_CR1_PS))

SMARTCARD_Request_Parameters

- #define: **SMARTCARD_RXDATA_FLUSH_REQUEST** ((*uint32_t*)USART_RQR_RXFRQ)
Receive Data flush Request
- #define: **SMARTCARD_TXDATA_FLUSH_REQUEST** ((*uint32_t*)USART_RQR_TXFRQ)

Transmit data flush Request

SMARTCARD_RTOR_BLEN_LSBPOS

- #define: ***SMARTCARD_RTOR_BLEN_LSB_POS ((uint32_t) 24)***

SMARTCARD_Rx_Inv

- #define: ***SMARTCARD_ADVFEATURE_RXINV_DISABLE ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_ADVFEATURE_RXINV_ENABLE ((uint32_t)USART_CR2_RXINV)***

SMARTCARD_Rx_Tx_Swap

- #define: ***SMARTCARD_ADVFEATURE_SWAP_DISABLE ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_ADVFEATURE_SWAP_ENABLE ((uint32_t)USART_CR2_SWAP)***

SMARTCARD_Stop_Bits

- #define: ***SMARTCARD_STOPBITS_1_5 ((uint32_t)(USART_CR2_STOP))***

SMARTCARD_Timeout_Enable

- #define: ***SMARTCARD_TIMEOUT_DISABLED ((uint32_t)0x00000000)***
- #define: ***SMARTCARD_TIMEOUT_ENABLED ((uint32_t)USART_CR2_RTOEN)***

SMARTCARD_Tx_Inv

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

-
- #define: **SMARTCARD_ADVFEATURE_TXINV_ENABLE**
((*uint32_t*)USART_CR2_TXINV)

SMARTCARD_Word_Length

- #define: **SMARTCARD_WORDLENGTH_9B** ((*uint32_t*)USART_CR1_M_0)

37 HAL SMARTCARD Extension Driver

37.1 SMARTCARDEEx Firmware driver introduction

37.2 SMARTCARDEEx Firmware driver API description

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

37.2.1 How to use this driver

The Extended SMARTCARD HAL driver can be used as follow:

1. After having configured the SMARTCARD basic features with `HAL_SMARTCARD_Init()`, then if required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsc AdvancedInit structure.

37.2.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- `HAL_SMARTCARDEEx_BlockLength_Config()` API allows to configure the Block Length on the fly
- `HAL_SMARTCARDEEx_TimeOut_Config()` API allows to configure the receiver timeout value on the fly
- `HAL_SMARTCARDEEx_EnableReceiverTimeOut()` API enables the receiver timeout feature
- `HAL_SMARTCARDEEx_DisableReceiverTimeOut()` API disables the receiver timeout feature
- `HAL_SMARTCARDEEx_BlockLength_Config\(\)`
- `HAL_SMARTCARDEEx_TimeOut_Config\(\)`
- `HAL_SMARTCARDEEx_EnableReceiverTimeOut\(\)`
- `HAL_SMARTCARDEEx_DisableReceiverTimeOut\(\)`

37.2.2.1 `HAL_SMARTCARDEEx_BlockLength_Config`

Function Name	<code>void HAL_SMARTCARDEEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsc, uint8_t BlockLength)</code>
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • BlockLength : SMARTCARD block length (8-bit long at most)
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

37.2.2.2 HAL_SMARTCARDEEx_TimeOut_Config

Function Name	<code>void HAL_SMARTCARDEEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsc, uint32_t TimeOutValue)</code>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle • TimeOutValue : receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

37.2.2.3 HAL_SMARTCARDEEx_EnableReceiverTimeOut

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARDEEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsc)</code>
Function Description	Enable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none"> • hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- None.

37.2.2.4 HAL_SMARTCARDEEx_DisableReceiverTimeOut

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARDEEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsc)</code>
Function Description	Disable the SMARTCARD receiver timeout feature.

Parameters	<ul style="list-style-type: none">• hsc : SMARTCARD handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

37.3 SMARTCARDEX Firmware driver defines

37.3.1 SMARTCARDEX

SMARTCARDEX

38 HAL SMBUS Generic Driver

38.1 SMBUS Firmware driver introduction

38.2 SMBUS Firmware driver registers structures

38.2.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the `stm32l0xx_hal_smbus.h`

Data Fields

- *uint32_t Timing*
- *uint32_t AnalogFilter*
- *uint32_t OwnAddress1*
- *uint32_t AddressingMode*
- *uint32_t DualAddressMode*
- *uint32_t OwnAddress2*
- *uint32_t OwnAddress2Masks*
- *uint32_t GeneralCallMode*
- *uint32_t NoStretchMode*
- *uint32_t PacketErrorCheckMode*
- *uint32_t PeripheralMode*
- *uint32_t SMBusTimeout*

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***
 - Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***
 - Specifies if Analog Filter is enable or not. This parameter can be a value of [*SMBUS_Analog_Filter*](#)
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***
 - Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***
 - Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of [*SMBUS_addressing_mode*](#)
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***
 - Specifies if dual addressing mode is selected. This parameter can be a value of [*SMBUS_dual_addressing_mode*](#)
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***
 - Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***

- Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of **SMBUS_own_address2_masks**
- **uint32_t SMBUS_InitTypeDef::GeneralCallMode**
 - Specifies if general call mode is selected. This parameter can be a value of **SMBUS_general_call_addressing_mode**
- **uint32_t SMBUS_InitTypeDef::NoStretchMode**
 - Specifies if nostretch mode is selected. This parameter can be a value of **SMBUS_nostretch_mode**
- **uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode**
 - Specifies if Packet Error Check mode is selected. This parameter can be a value of **SMBUS_packet_error_check_mode**
- **uint32_t SMBUS_InitTypeDef::PeripheralMode**
 - Specifies which mode of Peripheral is selected. This parameter can be a value of **SMBUS_peripheral_mode**
- **uint32_t SMBUS_InitTypeDef::SMBusTimeout**
 - Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

38.2.2 **SMBUS_HandleTypeDef**

SMBUS_HandleTypeDef is defined in the stm32l0xx_hal_smbus.h

Data Fields

- **I2C_TypeDef * Instance**
- **SMBUS_InitTypeDef Init**
- **uint8_t * pBuffPtr**
- **uint16_t XferSize**
- **IO uint16_t XferCount**
- **IO uint32_t XferOptions**
- **IO HAL_SMBUS_StateTypeDef PreviousState**
- **HAL_LockTypeDef Lock**
- **IO HAL_SMBUS_StateTypeDef State**
- **IO HAL_SMBUS_ErrorTypeDef ErrorCode**

Field Documentation

- **I2C_TypeDef* SMBUS_HandleTypeDef::Instance**
 - SMBUS registers base address
- **SMBUS_InitTypeDef SMBUS_HandleTypeDef::Init**
 - SMBUS communication parameters
- **uint8_t* SMBUS_HandleTypeDef::pBuffPtr**
 - Pointer to SMBUS transfer buffer
- **uint16_t SMBUS_HandleTypeDef::XferSize**
 - SMBUS transfer size
- **IO uint16_t SMBUS_HandleTypeDef::XferCount**
 - SMBUS transfer counter
- **IO uint32_t SMBUS_HandleTypeDef::XferOptions**

- SMBUS transfer options
- **`__IO HAL_SMBUS_StateTypeDef SMBUS_HandleTypeDef::PreviousState`**
 - SMBUS communication Previous state
- **`HAL_LockTypeDef SMBUS_HandleTypeDef::Lock`**
 - SMBUS locking object
- **`__IO HAL_SMBUS_StateTypeDef SMBUS_HandleTypeDef::State`**
 - SMBUS communication state
- **`__IO HAL_SMBUS_ErrorTypeDef SMBUS_HandleTypeDef::ErrorCode`**
 - SMBUS Error code

38.2.3 I2C_TypeDef

`I2C_TypeDef` is defined in the `stm32l051xx.h`

Data Fields

- **`__IO uint32_t CR1`**
- **`__IO uint32_t CR2`**
- **`__IO uint32_t OAR1`**
- **`__IO uint32_t OAR2`**
- **`__IO uint32_t TIMINGR`**
- **`__IO uint32_t TIMEOUTR`**
- **`__IO uint32_t ISR`**
- **`__IO uint32_t ICR`**
- **`__IO uint32_t PECR`**
- **`__IO uint32_t RXDR`**
- **`__IO uint32_t TXDR`**

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 1 register, Address offset: 0x08
- **`__IO uint32_t I2C_TypeDef::OAR2`**
 - I2C Own address 2 register, Address offset: 0x0C
- **`__IO uint32_t I2C_TypeDef::TIMINGR`**
 - I2C Timing register, Address offset: 0x10
- **`__IO uint32_t I2C_TypeDef::TIMEOUTR`**
 - I2C Timeout register, Address offset: 0x14
- **`__IO uint32_t I2C_TypeDef::ISR`**
 - I2C Interrupt and status register, Address offset: 0x18
- **`__IO uint32_t I2C_TypeDef::ICR`**
 - I2C Interrupt clear register, Address offset: 0x1C
- **`__IO uint32_t I2C_TypeDef::PECR`**
 - I2C PEC register, Address offset: 0x20
- **`__IO uint32_t I2C_TypeDef::RXDR`**

- I2C Receive data register, Address offset: 0x24
- ***__IO uint32_t I2C_TypeDef::TXDR***
 - I2C Transmit data register, Address offset: 0x28

38.3 SMBUS Firmware driver API description

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

38.3.1 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
 - Clock Timing
 - Bus Timeout
 - Analog Filter mode
 - Own Address 1
 - Addressing mode (Master, Slave)
 - Dual Addressing mode
 - Own Address 2
 - Own Address 2 Mask
 - General call mode
 - Nostretch mode
 - Packet Error Check mode
 - Peripheral mode
- Call the function HAL_SMBUS_DelInit() to restore the default configuration of the selected SMBUSx peripheral.
- [**HAL_SMBUS_Init\(\)**](#)
- [**HAL_SMBUS_DelInit\(\)**](#)
- [**HAL_SMBUS_MspInit\(\)**](#)
- [**HAL_SMBUS_MspDelInit\(\)**](#)

38.3.2 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is :
 - `HAL_SMBUS_IsDeviceReady()`
2. There is only one mode of transfer:
 - No-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. No-Blocking mode functions with Interrupt are :
 - `HAL_SMBUS_Master_Transmit_IT()`
 - `HAL_SMBUS_Master_Receive_IT()`
 - `HAL_SMBUS_Slave_Transmit_IT()`
 - `HAL_SMBUS_Slave_Receive_IT()`

- HAL_SMBUS_Slave_Listen_IT()
 - HAL_SMBUS_EnableAlert_IT()
 - HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
- HAL_SMBUS_MasterTxCpltCallback()
 - HAL_SMBUS_MasterRxCpltCallback()
 - HAL_SMBUS_SlaveTxCpltCallback()
 - HAL_SMBUS_SlaveRxCpltCallback()
 - HAL_SMBUS_SlaveAddrCallback()
 - HAL_SMBUS_SlaveListenCpltCallback()
 - HAL_SMBUS_ErrorCallback()
 - ***HAL_SMBUS_Master_Transmit_IT()***
 - ***HAL_SMBUS_Master_Receive_IT()***
 - ***HAL_SMBUS_Master_Abort_IT()***
 - ***HAL_SMBUS_Slave_Transmit_IT()***
 - ***HAL_SMBUS_Slave_Receive_IT()***
 - ***HAL_SMBUS_Slave_Listen_IT()***
 - ***HAL_SMBUS_EnableAlert_IT()***
 - ***HAL_SMBUS_DisableAlert_IT()***
 - ***HAL_SMBUS_IsDeviceReady()***
 - ***HAL_SMBUS_EV_IRQHandler()***
 - ***HAL_SMBUS_ER_IRQHandler()***
 - ***HAL_SMBUS_MasterTxCpltCallback()***
 - ***HAL_SMBUS_MasterRxCpltCallback()***
 - ***HAL_SMBUS_SlaveTxCpltCallback()***
 - ***HAL_SMBUS_SlaveRxCpltCallback()***
 - ***HAL_SMBUS_SlaveAddrCallback()***
 - ***HAL_SMBUS_SlaveListenCpltCallback()***
 - ***HAL_SMBUS_ErrorCallback()***

38.3.3 Peripheral State and Errors functions

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

- ***HAL_SMBUS_GetState()***
- ***HAL_SMBUS_GetError()***

38.3.3.1 HAL_SMBUS_Init

Function Name	<i>HAL_StatusTypeDef HAL_SMBUS_Init (</i> <i>SMBUS_HandleTypeDef * hsmbus)</i>
Function Description	Initializes the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • <i>hsmbus</i> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• <i>HAL status</i>
Notes	• None.

38.3.3.2 HAL_SMBUS_DelInit

Function Name	HAL_StatusTypeDef HAL_SMBUS_DelInit (SMBUS_HandleTypeDef * hsmbus)
Function Description	DeInitializes the SMBUS peripheral.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.3 HAL_SMBUS_MspInit

Function Name	void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)
Function Description	SMBUS MSP Init.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

38.3.3.4 HAL_SMBUS_MspDelInit

Function Name	void HAL_SMBUS_MspDelInit (SMBUS_HandleTypeDef * hsmbus)
Function Description	SMBUS MSP DelInit.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure

that contains the configuration information for the specified SMBUS.

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

38.3.3.5 HAL_SMBUS_Master_Transmit_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (</code> SMBUS_HandleTypeDef * <code>hsmbus</code> , <code>uint16_t DevAddress</code> , <code>uint8_t * pData</code> , <code>uint16_t Size</code> , <code>uint32_t XferOptions</code>)
Function Description	Transmit in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. DevAddress : Target device address pData : Pointer to data buffer Size : Amount of data to be sent XferOptions : Options of Transfer, value of SMBUS_XferOptions_definition
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

38.3.3.6 HAL_SMBUS_Master_Receive_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (</code> SMBUS_HandleTypeDef * <code>hsmbus</code> , <code>uint16_t DevAddress</code> , <code>uint8_t * pData</code> , <code>uint16_t Size</code> , <code>uint32_t XferOptions</code>)
Function Description	Receive in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. DevAddress : Target device address pData : Pointer to data buffer Size : Amount of data to be sent

	<ul style="list-style-type: none"> • XferOptions : Options of Transfer, value of SMBUS_XferOptions_definition
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.7 HAL_SMBUS_Master_Abort_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)
Function Description	Abort a master/host SMBUS process communication with Interrupt.
Parameters	<ul style="list-style-type: none"> • hsmbus : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • DevAddress : Target device address
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • : This abort can be called only if state is ready

38.3.3.8 HAL_SMBUS_Slave_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
Function Description	Transmit in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hsmbus : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • pData : Pointer to data buffer • Size : Amount of data to be sent • XferOptions : Options of Transfer, value of SMBUS_XferOptions_definition
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.9 HAL_SMBUS_Slave_Receive_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</code>
Function Description	Receive in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • pData : Pointer to data buffer • Size : Amount of data to be sent • XferOptions : Options of Transfer, value of SMBUS_XferOptions_definition
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.10 HAL_SMBUS_Slave_Listen_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Slave_Listen_IT (SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	This function enable the Address listen mode in Slave mode.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.11 HAL_SMBUS_EnableAlert_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)
Function Description	Enable SMBUS alert.
Parameters	<ul style="list-style-type: none"> • hsmbus : : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.12 HAL_SMBUS_DisableAlert_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)
Function Description	Disable SMBUS alert.
Parameters	<ul style="list-style-type: none"> • hsmbus : : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

38.3.3.13 HAL_SMBUS_IsDeviceReady

Function Name	HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • DevAddress : Target device address • Trials : Number of trials • Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used with Memory devices

38.3.3.14 HAL_SMBUS_EV_IRQHandler

Function Name	<code>void HAL_SMBUS_EV_IRQHandler (<i>SMBUS_HandleTypeDef</i> * hsmbus)</code>
Function Description	This function handles SMBUS event interrupt request.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a <i>SMBUS_HandleTypeDef</i> structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

38.3.3.15 HAL_SMBUS_ER_IRQHandler

Function Name	<code>void HAL_SMBUS_ER_IRQHandler (<i>SMBUS_HandleTypeDef</i> * hsmbus)</code>
Function Description	This function handles SMBUS error interrupt request.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a <i>SMBUS_HandleTypeDef</i> structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

38.3.3.16 HAL_SMBUS_MasterTxCpltCallback

Function Name	<code>void HAL_SMBUS_MasterTxCpltCallback (<i>SMBUS_HandleTypeDef</i> * hsmbus)</code>
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a <i>SMBUS_HandleTypeDef</i> structure

that contains the configuration information for the specified SMBUS.

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

38.3.3.17 HAL_SMBUS_MasterRxCpltCallback

Function Name	void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

38.3.3.18 HAL_SMBUS_SlaveTxCpltCallback

Function Name	void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

38.3.3.19 HAL_SMBUS_SlaveRxCpltCallback

Function Name	void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

38.3.3.20 HAL_SMBUS_SlaveAddrCallback

Function Name	void HAL_SMBUS_SlaveAddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)
Function Description	Slave Address Match callbacks.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • TransferDirection : Master request Transfer Direction (Write/Read) • AddrMatchCode : Address Match Code
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

38.3.3.21 HAL_SMBUS_SlaveListenCpltCallback

Function Name	void HAL_SMBUS_SlaveListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Listen Complete callbacks.
Parameters	<ul style="list-style-type: none"> • hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

38.3.3.22 HAL_SMBUS_ErrorCallback

Function Name	<code>void HAL_SMBUS_ErrorCallback (<i>SMBUS_HandleTypeDef</i> * hsmbus)</code>
Function Description	SMBUS error callbacks.
Parameters	<ul style="list-style-type: none">• hsmbus : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

38.3.3.23 HAL_SMBUS_GetState

Function Name	<code>HAL_SMBUS_StateTypeDef HAL_SMBUS_GetState (<i>SMBUS_HandleTypeDef</i> * hsmbus)</code>
Function Description	Returns the SMBUS state.
Parameters	<ul style="list-style-type: none">• hsmbus : : SMBUS handle
Return values	<ul style="list-style-type: none">• HAL state
Notes	<ul style="list-style-type: none">• None.

38.3.3.24 HAL_SMBUS_GetError

Function Name	<code>uint32_t HAL_SMBUS_GetError (<i>SMBUS_HandleTypeDef</i> * hsmbus)</code>
Function Description	Return the SMBUS error code.
Parameters	<ul style="list-style-type: none">• hsmbus : : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values	• SMBUS Error Code
Notes	• None.

38.4 SMBUS Firmware driver defines

38.4.1 SMBUS

SMBUS

SMBUS_addressing_mode

- #define: **SMBUS_ADDRESSINGMODE_7BIT** ((uint32_t)0x00000001)

- #define: **SMBUS_ADDRESSINGMODE_10BIT** ((uint32_t)0x00000002)

SMBUS_Analog_Filter

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

- #define: **SMBUS_ANALOGFILTER_DISABLED I2C_CR1_ANFOFF**

SMBUS_dual_addressing_mode

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

- #define: **SMBUS_DUALADDRESS_ENABLED I2C_OAR2_OA2EN**

SMBUS_Flag_definition

- #define: **SMBUS_FLAG_TXE I2C_ISR_TXE**

- #define: **SMBUS_FLAG_RXIS I2C_ISR_RXIS**

- #define: **SMBUS_FLAG_RXNE I2C_ISR_RXNE**
- #define: **SMBUS_FLAG_ADDR I2C_ISR_ADDR**
- #define: **SMBUS_FLAG_AF I2C_ISR_NACKF**
- #define: **SMBUS_FLAG_STOPF I2C_ISR_STOPF**
- #define: **SMBUS_FLAG_TC I2C_ISR_TC**
- #define: **SMBUS_FLAG_TCR I2C_ISR_TCR**
- #define: **SMBUS_FLAG_BERR I2C_ISR_BERR**
- #define: **SMBUS_FLAG_ARLO I2C_ISR_ARLO**
- #define: **SMBUS_FLAG_OVR I2C_ISR_OVR**
- #define: **SMBUS_FLAG_PECERR I2C_ISR_PECERR**
- #define: **SMBUS_FLAG_TIMEOUT I2C_ISR_TIMEOUT**
- #define: **SMBUS_FLAG_ALERT I2C_ISR_ALERT**

- #define: **SMBUS_FLAG_BUSY I2C_ISR_BUSY**

- #define: **SMBUS_FLAG_DIR I2C_ISR_DIR**

SMBUS_general_call_addressing_mode

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

- #define: **SMBUS_GENERALCALL_ENABLED I2C_CR1_GCEN**

SMBUS_Interrupt_configuration_definition

- #define: **SMBUS_IT_ERRI I2C_CR1_ERRIE**

- #define: **SMBUS_IT_TCI I2C_CR1_TCIE**

- #define: **SMBUS_IT_STOPI I2C_CR1_STOPIE**

- #define: **SMBUS_IT_NACKI I2C_CR1_NACKIE**

- #define: **SMBUS_IT_ADDRI I2C_CR1_ADDRIE**

- #define: **SMBUS_IT_RXI I2C_CR1_RXIE**

- #define: **SMBUS_IT_TXI I2C_CR1_TXIE**

- #define: **SMBUS_IT_TX** (**SMBUS_IT_ERRI** | **SMBUS_IT_TCI** | **SMBUS_IT_STOPI** |
SMBUS_IT_NACKI | **SMBUS_IT_TXI**)
- #define: **SMBUS_IT_RX** (**SMBUS_IT_ERRI** | **SMBUS_IT_TCI** | **SMBUS_IT_NACKI** |
SMBUS_IT_RXI)
- #define: **SMBUS_IT_ALERT** (**SMBUS_IT_ERRI**)
- #define: **SMBUS_IT_ADDR** (**SMBUS_IT_ADDRI** | **SMBUS_IT_STOPI** |
SMBUS_IT_NACKI)

SMBUS_nostretch_mode

- #define: **SMBUS_NOSTRETCH_DISABLED** ((**uint32_t**)0x00000000)
- #define: **SMBUS_NOSTRETCH_ENABLED I2C_CR1_NOSTRETCH**

SMBUS_own_address2_masks

- #define: **SMBUS_OA2_NOMASK** ((**uint8_t**)0x00)
- #define: **SMBUS_OA2_MASK01** ((**uint8_t**)0x01)
- #define: **SMBUS_OA2_MASK02** ((**uint8_t**)0x02)
- #define: **SMBUS_OA2_MASK03** ((**uint8_t**)0x03)
- #define: **SMBUS_OA2_MASK04** ((**uint8_t**)0x04)

- #define: **SMBUS_OA2_MASK05** ((*uint8_t*)0x05)

- #define: **SMBUS_OA2_MASK06** ((*uint8_t*)0x06)

- #define: **SMBUS_OA2_MASK07** ((*uint8_t*)0x07)

SMBUS_packet_error_check_mode

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

- #define: **SMBUS_PEC_ENABLED I2C_CR1_PECEN**

SMBUS_peripheral_mode

- #define: **SMBUS_PERIPHERAL_MODE_SMBUS_HOST** (*uint32_t*)(I2C_CR1_SMBHEN)

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

- #define: **SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP** (*uint32_t*)(I2C_CR1_SMBDEN)

SMBUS_ReloadEndMode_definition

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

- #define: **SMBUS_RELOAD_MODE I2C_CR2_RELOAD**

- #define: **SMBUS_AUTOEND_MODE I2C_CR2_AUTOEND**

- #define: **SMBUS_SENDPEC_MODE I2C_CR2_PECBYTE**
-
- SMBUS_StartStopMode_definition***
- #define: **SMBUS_NO_STARTSTOP ((uint32_t)0x00000000)**
 - #define: **SMBUS_GENERATE_STOP I2C_CR2_STOP**
 - #define: **SMBUS_GENERATE_START_READ (uint32_t)(I2C_CR2_START | I2C_CR2_RD_WRN)**
 - #define: **SMBUS_GENERATE_START_WRITE I2C_CR2_START**
-
- SMBUS_XferOptions_definition***
- #define: **SMBUS_FIRST_FRAME ((uint32_t)(SMBUS_SOFTEND_MODE))**
 - #define: **SMBUS_NEXT_FRAME ((uint32_t)(SMBUS_RELOAD_MODE | SMBUS_SOFTEND_MODE))**
 - #define: **SMBUS_FIRST_AND_LAST_FRAME_NO_PEC
SMBUS_AUTOEND_MODE**
 - #define: **SMBUS_LAST_FRAME_NO_PEC SMBUS_AUTOEND_MODE**
 - #define: **SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC
((uint32_t)(SMBUS_AUTOEND_MODE | SMBUS_SENDPEC_MODE))**

-
- #define: **SMBUS_LAST_FRAME_WITH_PEC**
((uint32_t)(SMBUS_AUTOEND_MODE | SMBUS_SENDPEC_MODE))

39 HAL SPI Generic Driver

39.1 SPI Firmware driver introduction

39.2 SPI Firmware driver registers structures

39.2.1 SPI_HandleTypeDef

`SPI_HandleTypeDef` is defined in the `stm32l0xx_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`

39.2.2 SPI_InitTypeDef

SPI_InitTypeDef is defined in the `stm32l0xx_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

39.2.3 SPI_TypeDef

SPI_TypeDef is defined in the stm32l051xx.h

Data Fields

- *__IO uint16_t CR1*
- *uint16_t RESERVED0*
- *__IO uint16_t CR2*
- *uint16_t RESERVED1*
- *__IO uint16_t SR*
- *uint16_t RESERVED2*
- *__IO uint16_t DR*
- *uint16_t RESERVED3*
- *__IO uint16_t CRCPR*
- *uint16_t RESERVED4*
- *__IO uint16_t RXCRCR*
- *uint16_t RESERVED5*
- *__IO uint16_t TXCRCR*
- *uint16_t RESERVED6*
- *__IO uint16_t I2SCFGR*
- *uint16_t RESERVED7*
- *__IO uint16_t I2SPR*
- *uint16_t RESERVED8*

Field Documentation

- *__IO uint16_t SPI_TypeDef::CR1*
 - SPI Control register 1 (not used in I2S mode), Address offset: 0x00
- *uint16_t SPI_TypeDef::RESERVED0*
 - Reserved, 0x02
- *__IO uint16_t SPI_TypeDef::CR2*
 - SPI Control register 2, Address offset: 0x04
- *uint16_t SPI_TypeDef::RESERVED1*
 - Reserved, 0x06
- *__IO uint16_t SPI_TypeDef::SR*
 - SPI Status register, Address offset: 0x08
- *uint16_t SPI_TypeDef::RESERVED2*
 - Reserved, 0x0A
- *__IO uint16_t SPI_TypeDef::DR*
 - SPI data register, Address offset: 0x0C
- *uint16_t SPI_TypeDef::RESERVED3*
 - Reserved, 0x0E
- *__IO uint16_t SPI_TypeDef::CRCPR*
 - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10
- *uint16_t SPI_TypeDef::RESERVED4*
 - Reserved, 0x12

- **`__IO uint16_t SPI_TypeDef::RXCRCR`**
 - SPI Rx CRC register (not used in I2S mode), Address offset: 0x14
- **`uint16_t SPI_TypeDef::RESERVED5`**
 - Reserved, 0x16
- **`__IO uint16_t SPI_TypeDef::TXCRCR`**
 - SPI Tx CRC register (not used in I2S mode), Address offset: 0x18
- **`uint16_t SPI_TypeDef::RESERVED6`**
 - Reserved, 0x1A
- **`__IO uint16_t SPI_TypeDef::I2SCFGR`**
 - SPI_I2S configuration register, Address offset: 0x1C
- **`uint16_t SPI_TypeDef::RESERVED7`**
 - Reserved, 0x1E
- **`__IO uint16_t SPI_TypeDef::I2SPR`**
 - SPI_I2S prescaler register, Address offset: 0x20
- **`uint16_t SPI_TypeDef::RESERVED8`**
 - Reserved, 0x22

39.3 SPI Firmware driver API description

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

39.3.1 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 implement 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 initialized 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 customized HAL_SPI_MspInit(&hspi) API.



The max SPI frequency depend on SPI data size (4bits, 5bits,..., 8bits,...15bits, 16bits), SPI mode(2 Lines fullduplex, 2 lines RxOnly, 1 line TX/RX) and Process mode (Polling, IT, DMA).

- TX/RX processes are HAL_SPI_TransmitReceive(), HAL_SPI_TransmitReceive_IT() and HAL_SPI_TransmitReceive_DMA()
- RX processes are HAL_SPI_Receive(), HAL_SPI_Receive_IT() and HAL_SPI_Receive_DMA()
- TX processes are HAL_SPI_Transmit(), HAL_SPI_Transmit_IT() and HAL_SPI_Transmit_DMA()

39.3.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must Implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
 - Mode
 - Direction
 - Data Size
 - Clock Polarity and Phase
 - NSS Management
 - BaudRate Prescaler
 - FirstBit
 - TIMode
 - CRC Calculation
 - CRC Polynomial if CRC enabled
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.
- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***

39.3.3 IO operation functions

The SPI supports master and slave mode :

1. There are two mode 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 SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The

- HAL_SPI_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's 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. No-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_TransmitReceive_DMA() in full duplex mode
 5. A set of Transfer Complete Callbacks are provided in No_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()***

39.3.4 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()***

39.3.4.1 HAL_SPI_Init

Function Name	HAL_StatusTypeDef HAL_SPI_Init (<i>SPI_HandleTypeDef</i> * hspi)
---------------	--

Function Description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.2 HAL_SPI_DelInit

Function Name	HAL_StatusTypeDef HAL_SPI_DelInit (SPI_HandleTypeDef * hspi)
Function Description	Deinitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.3 HAL_SPI_MspInit

Function Name	void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

39.3.4.4 HAL_SPI_MspDelInit

Function Name	void HAL_SPI_MspDelinit (<i>SPI_HandleTypeDef</i> * hspi)
Function Description	SPI MSP Delinit.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

39.3.4.5 HAL_SPI_Transmit

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit (<i>SPI_HandleTypeDef</i> * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.6 HAL_SPI_Receive

Function Name	HAL_StatusTypeDef HAL_SPI_Receive (<i>SPI_HandleTypeDef</i> * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.7 HAL_SPI_TransmitReceive

Function Name	<code>HAL_StatusTypeDef HAL_SPI_TransmitReceive (</code> <code>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.8 HAL_SPI_Transmit_IT

Function Name	<code>HAL_StatusTypeDef HAL_SPI_Transmit_IT (</code> <code>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</code>
Function Description	Transmit an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.9 HAL_SPI_Receive_IT

Function Name	<code>HAL_StatusTypeDef HAL_SPI_Receive_IT (</code>
---------------	---

SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function Description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.10 HAL_SPI_TransmitReceive_IT

Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (<i>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</i>
Function Description	Transmit and Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.11 HAL_SPI_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit_DMA (<i>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</i>
Function Description	Transmit an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

39.3.4.12 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	<ul style="list-style-type: none"> hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. pData : pointer to data buffer
Parameters	<ul style="list-style-type: none"> Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the CRC feature is enabled the pData Length must be Size + 1.

39.3.4.13 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the CRC feature is enabled the pRxData Length must be Size + 1

39.3.4.14 HAL_SPI_IRQHandler

Function Name	void HAL_SPI_IRQHandler (<i>SPI_HandleTypeDef</i> * hspi)
Function Description	This function handles SPI interrupt request.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

39.3.4.15 HAL_SPI_TxCpltCallback

Function Name	void HAL_SPI_TxCpltCallback (<i>SPI_HandleTypeDef</i> * hspi)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

39.3.4.16 HAL_SPI_RxCpltCallback

Function Name	void HAL_SPI_RxCpltCallback (<i>SPI_HandleTypeDef</i> * hspi)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

39.3.4.17 HAL_SPI_TxRxCpltCallback

Function Name	void HAL_SPI_TxRxCpltCallback (<i>SPI_HandleTypeDef</i> * <i>hspi</i>)
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

39.3.4.18 HAL_SPI_ErrorCallback

Function Name	void HAL_SPI_ErrorCallback (<i>SPI_HandleTypeDef</i> * <i>hspi</i>)
Function Description	SPI error callbacks.
Parameters	<ul style="list-style-type: none">• hspi : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

39.3.4.19 HAL_SPI_GetState

Function Name	HAL_SPI_StateTypeDef HAL_SPI_GetState (<i>SPI_HandleTypeDef</i> * <i>hspi</i>)
Function Description	Return the SPI state.
Parameters	<ul style="list-style-type: none">• hspi : : SPI handle
Return values	<ul style="list-style-type: none">• SPI state
Notes	<ul style="list-style-type: none">• None.

39.3.4.20 HAL_SPI_GetError

Function Name	HAL_SPI_ErrorTypeDef HAL_SPI_GetError (SPI_HandleTypeDef * hspi)
Function Description	Return the SPI error code.
Parameters	<ul style="list-style-type: none"> • hspi : : SPI handle
Return values	<ul style="list-style-type: none"> • SPI Error Code
Notes	<ul style="list-style-type: none"> • None.

39.4 SPI Firmware driver defines

39.4.1 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***

40 HAL TIM Generic Driver

40.1 TIM Firmware driver introduction

40.2 TIM Firmware driver registers structures

40.2.1 TIM_HandleTypeDef

TIM_HandleTypeDef is defined in the `stm32l0xx_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

40.2.2 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the `stm32l0xx_hal_tim.h`

Data Fields

- *uint32_t Prescaler*
- *uint32_t CounterMode*
- *uint32_t Period*
- *uint32_t ClockDivision*

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**](#)

40.2.3 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the `stm32l0xx_hal_tim.h`

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCFastMode***

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::OCFastMode***
 - Specifies the Fast mode state. This parameter can be a value of [**TIM_Output_Fast_State**](#)

40.2.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the `stm32l0xx_hal_tim.h`

Data Fields

- *uint32_t IC_Polarity*
- *uint32_t IC_Selection*
- *uint32_t IC_Prescaler*
- *uint32_t IC_Filter*

Field Documentation

- *uint32_t TIM_IC_InitTypeDef::IC_Polarity*
 - Specifies the active edge of the input signal. This parameter can be a value of [TIM_Input_Capture_Polarity](#)
- *uint32_t TIM_IC_InitTypeDef::IC_Selection*
 - Specifies the input. This parameter can be a value of [TIM_Input_Capture_Selection](#)
- *uint32_t TIM_IC_InitTypeDef::IC_Prescaler*
 - Specifies the Input Capture Prescaler. This parameter can be a value of [TIM_Input_Capture_Prescaler](#)
- *uint32_t TIM_IC_InitTypeDef::IC_Filter*
 - Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

40.2.5 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the `stm32l0xx_hal_tim.h`

Data Fields

- *uint32_t OCMode*
- *uint32_t Pulse*
- *uint32_t OCPolarity*
- *uint32_t IC_Polarity*
- *uint32_t IC_Selection*
- *uint32_t IC_Filter*

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::IC_Polarity*

- 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

40.2.6 **TIM_ClockConfigTypeDef**

TIM_ClockConfigTypeDef is defined in the `stm32l0xx_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

40.2.7 **TIM_ClearInputConfigTypeDef**

TIM_ClearInputConfigTypeDef is defined in the `stm32l0xx_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

40.2.8 **TIM_SlaveConfigTypeDef**

TIM_SlaveConfigTypeDef is defined in the *stm32l0xx_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

40.2.9 **TIM_Encoder_InitTypeDef**

TIM_Encoder_InitTypeDef is defined in the *stm32l0xx_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

40.2.10 TIM_TypeDef

`TIM_TypeDef` is defined in the `stm32l051xx.h`

Data Fields

- `__IO uint16_t CR1`
- `uint16_t RESERVED0`
- `__IO uint16_t CR2`
- `uint16_t RESERVED1`
- `__IO uint16_t SMCR`
- `uint16_t RESERVED2`

- `__IO uint16_t DIER`
- `uint16_t RESERVED3`
- `__IO uint16_t SR`
- `uint16_t RESERVED4`
- `__IO uint16_t EGR`
- `uint16_t RESERVED5`
- `__IO uint16_t CCMR1`
- `uint16_t RESERVED6`
- `__IO uint16_t CCMR2`
- `uint16_t RESERVED7`
- `__IO uint16_t CCER`
- `uint16_t RESERVED8`
- `__IO uint32_t CNT`
- `__IO uint16_t PSC`
- `uint16_t RESERVED10`
- `__IO uint32_t ARR`
- `__IO uint16_t RCR`
- `uint16_t RESERVED12`
- `__IO uint32_t CCR1`
- `__IO uint32_t CCR2`
- `__IO uint32_t CCR3`
- `__IO uint32_t CCR4`
- `__IO uint16_t BDTR`
- `uint16_t RESERVED17`
- `__IO uint16_t DCR`
- `uint16_t RESERVED18`
- `__IO uint16_t DMAR`
- `uint16_t RESERVED19`
- `__IO uint16_t OR`
- `uint16_t RESERVED20`

Field Documentation

- `__IO uint16_t TIM_TypeDef::CR1`
 - TIM control register 1, Address offset: 0x00
- `uint16_t TIM_TypeDef::RESERVED0`
 - Reserved, 0x02
- `__IO uint16_t TIM_TypeDef::CR2`
 - TIM control register 2, Address offset: 0x04
- `uint16_t TIM_TypeDef::RESERVED1`
 - Reserved, 0x06
- `__IO uint16_t TIM_TypeDef::SMCR`
 - TIM slave Mode Control register, Address offset: 0x08
- `uint16_t TIM_TypeDef::RESERVED2`
 - Reserved, 0x0A
- `__IO uint16_t TIM_TypeDef::DIER`
 - TIM DMA/interrupt enable register, Address offset: 0x0C
- `uint16_t TIM_TypeDef::RESERVED3`
 - Reserved, 0x0E
- `__IO uint16_t TIM_TypeDef::SR`
 - TIM status register, Address offset: 0x10

- ***uint16_t TIM_TypeDef::RESERVED4***
 - Reserved, 0x12
- ***_IO uint16_t TIM_TypeDef::EGR***
 - TIM event generation register, Address offset: 0x14
- ***uint16_t TIM_TypeDef::RESERVED5***
 - Reserved, 0x16
- ***_IO uint16_t TIM_TypeDef::CCMR1***
 - TIM capture/compare mode register 1, Address offset: 0x18
- ***uint16_t TIM_TypeDef::RESERVED6***
 - Reserved, 0x1A
- ***_IO uint16_t TIM_TypeDef::CCMR2***
 - TIM capture/compare mode register 2, Address offset: 0x1C
- ***uint16_t TIM_TypeDef::RESERVED7***
 - Reserved, 0x1E
- ***_IO uint16_t TIM_TypeDef::CCER***
 - TIM capture/compare enable register, Address offset: 0x20
- ***uint16_t TIM_TypeDef::RESERVED8***
 - Reserved, 0x22
- ***_IO uint32_t TIM_TypeDef::CNT***
 - TIM counter register, Address offset: 0x24
- ***_IO uint16_t TIM_TypeDef::PSC***
 - TIM prescaler register, Address offset: 0x28
- ***uint16_t TIM_TypeDef::RESERVED10***
 - Reserved, 0x2A
- ***_IO uint32_t TIM_TypeDef::ARR***
 - TIM auto-reload register, Address offset: 0x2C
- ***_IO uint16_t TIM_TypeDef::RCR***
 - TIM repetition counter register, Address offset: 0x30
- ***uint16_t TIM_TypeDef::RESERVED12***
 - Reserved, 0x32
- ***_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 uint16_t TIM_TypeDef::BDTR***
 - TIM break and dead-time register, Address offset: 0x44
- ***uint16_t TIM_TypeDef::RESERVED17***
 - Reserved, 0x26
- ***_IO uint16_t TIM_TypeDef::DCR***
 - TIM DMA control register, Address offset: 0x48
- ***uint16_t TIM_TypeDef::RESERVED18***
 - Reserved, 0x4A
- ***_IO uint16_t TIM_TypeDef::DMAR***
 - TIM DMA address for full transfer register, Address offset: 0x4C
- ***uint16_t TIM_TypeDef::RESERVED19***
 - Reserved, 0x4E
- ***_IO uint16_t TIM_TypeDef::OR***
 - TIM option register, Address offset: 0x50

-
- ***uint16_t TIM_TypeDef::RESERVED20***
 - Reserved, 0x52

40.3 TIM Firmware driver API description

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

40.3.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
 - Input Capture
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

40.3.2 How to use this driver

1. Enable the TIM interface clock using `__TIMx_CLK_ENABLE()`;
2. 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: `HAL_TIM_Base_Start()`, `HAL_TIM_Base_Start_DMA()`, `HAL_TIM_Base_Start_IT()`, `HAL_TIM_OC_Start()`, `HAL_TIM_OC_Start_DMA()`, `HAL_TIM_OC_Start_IT()`, `HAL_TIM_IC_Start()`, `HAL_TIM_IC_Start_DMA()`, `HAL_TIM_IC_Start_IT()`, `HAL_TIM_PWM_Start()`,

- `HAL_TIM_PWM_Start_DMA()`, `HAL_TIM_PWM_Start_IT()`,
`HAL_TIM_OnePulse_Start()`, `HAL_TIM_OnePulse_Start_IT()`,
`HAL_TIM_Encoder_Start()`, `HAL_TIM_Encoder_Start_DMA()` or
`HAL_TIM_Encoder_Start_IT()`
6. The DMA Burst is managed with the two following functions:
`HAL_TIM_DMABurst_WriteStart()` `HAL_TIM_DMABurst_ReadStart()`

40.3.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TIM.
- De-initialize the TIM.
- `HAL_TIM_Base_Init()`
- `HAL_TIM_OC_Init()`
- `HAL_TIM_PWM_Init()`
- `HAL_TIM_IC_Init()`
- `HAL_TIM_OnePulse_Init()`
- `HAL_TIM_Encoder_Init()`
- `HAL_TIM_Base_DelInit()`
- `HAL_TIM_OC_DelInit()`
- `HAL_TIM_PWM_DelInit()`
- `HAL_TIM_IC_DelInit()`
- `HAL_TIM_OnePulse_DelInit()`
- `HAL_TIM_Encoder_DelInit()`
- `HAL_TIM_Base_MspInit()`
- `HAL_TIM_OC_MspInit()`
- `HAL_TIM_PWM_MspInit()`
- `HAL_TIM_IC_MspInit()`
- `HAL_TIM_OnePulse_MspInit()`
- `HAL_TIM_Encoder_MspInit()`
- `HAL_TIM_Base_MspDelInit()`
- `HAL_TIM_OC_MspDelInit()`
- `HAL_TIM_PWM_MspDelInit()`
- `HAL_TIM_IC_MspDelInit()`
- `HAL_TIM_OnePulse_MspDelInit()`
- `HAL_TIM_Encoder_MspDelInit()`

40.3.4 IO operation functions

This section provides functions allowing to:

- 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.
- Start the Output Compare/PWM.
- Stop the Output Compare/PWM.
- Start the Output Compare/PWM and enable interrupts.
- Stop the Output Compare/PWM and disable interrupts.

- Start the Output Compare/PWM and enable DMA transfers.
- Stop the Output Compare/PWM and disable DMA transfers.
- Start the Input Capture measurement.
- Stop the Input Capture.
- Start the Input Capture and enable interrupts.
- Stop the Input Capture and disable interrupts.
- Start the Input Capture and enable DMA transfers.
- Stop the Input Capture and disable DMA transfers.
- Start the One Pulse generation.
- Stop the One Pulse.
- Start the One Pulse and enable interrupts.
- Stop the One Pulse and disable interrupts.
- Start the Encoder Interface.
- Stop the Encoder Interface.
- Start the Encoder Interface and enable interrupts.
- Stop the Encoder Interface and disable interrupts.
- Start the Encoder Interface and enable DMA transfers.
- Stop the Encoder Interface and disable DMA transfers.
- 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.
- Handle TIM interrupt request.
- *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()*
- *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()*
- *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()*
- *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()*
- *HAL_TIM_OnePulse_Start()*
- *HAL_TIM_OnePulse_Stop()*
- *HAL_TIM_OnePulse_Start_IT()*

- `HAL_TIM_OnePulse_Stop_IT()`
- `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()`

40.3.5 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 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()`

40.3.6 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()`

40.3.7 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()`

40.3.8 IRQ handler management

This section provides Timer IRQ handler function.

- `HAL_TIM_IRQHandler()`
- `HAL_TIM_DMAError()`
- `HAL_TIM_DMADelayPulseCplt()`
- `HAL_TIM_DMACaptureCplt()`

40.3.8.1 HAL_TIM_Base_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the <code>TIM_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • <code>htim</code> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.2 HAL_TIM_OC_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Output Compare according to the specified parameters in the <code>TIM_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • <code>htim</code> : TIM Output Compare handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.3 HAL_TIM_PWM_Init

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Init (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.4 HAL_TIM_IC_Init

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Init (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim : TIM Input Capture handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.5 HAL_TIM_OnePulse_Init

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Init (<i>TIM_HandleTypeDef</i> * htim, uint32_t OnePulseMode)
Function Description	Initializes the TIM One Pulse Time Base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.

Parameters	<ul style="list-style-type: none"> • htim : TIM OnePulse handle • OnePulseMode : Select the One pulse mode. This parameter can be one of the following values: <ul style="list-style-type: none"> – TIM_OPmode_SINGLE : Only one pulse will be generated. – TIM_OPmode_REPEATITIVE : Repetitive pulses wil be generated.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.6 HAL_TIM_Encoder_Init

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Init (<i>TIM_HandleTypeDef</i> * htim, <i>TIM_Encoder_InitTypeDef</i> * sConfig)
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim : TIM Encoder Interface handle • sConfig : TIM Encoder Interface configuration structure
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.7 HAL_TIM_Base_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Base_DeInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.8 HAL_TIM_OC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_OC_DeInit (<i>TIM_HandleTypeDef * htim</i>)
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none">• htim : TIM Output Compare handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.9 HAL_TIM_PWM_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_DeInit (<i>TIM_HandleTypeDef * htim</i>)
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.10 HAL_TIM_IC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_IC_DeInit (<i>TIM_HandleTypeDef * htim</i>)
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none">• htim : TIM Input Capture handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.11 HAL_TIM_OnePulse_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (<i>TIM_HandleTypeDef * htim</i>)
Function Description	DeInitializes the TIM One Pulse.
Parameters	<ul style="list-style-type: none"> • htim : TIM One Pulse handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.12 HAL_TIM_Encoder_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (<i>TIM_HandleTypeDef * htim</i>)
Function Description	DeInitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> • htim : TIM Encoder handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.13 HAL_TIM_Base_MspInit

Function Name	void HAL_TIM_Base_MspInit (<i>TIM_HandleTypeDef * htim</i>)
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.14 HAL_TIM_OC_MspInit

Function Name	void HAL_TIM_OC_MspInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

40.3.8.15 HAL_TIM_PWM_MspInit

Function Name	void HAL_TIM_PWM_MspInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM PWM MSP.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

40.3.8.16 HAL_TIM_IC_MspInit

Function Name	void HAL_TIM_IC_MspInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM INput Capture MSP.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

40.3.8.17 HAL_TIM_OnePulse_MspInit

Function Name	void HAL_TIM_OnePulse_MspInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.18 HAL_TIM_Encoder_MspInit

Function Name	void HAL_TIM_Encoder_MspInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.19 HAL_TIM_Base_MspDeInit

Function Name	void HAL_TIM_Base_MspDeInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Deinitializes TIM Base MSP.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.20 HAL_TIM_OC_MspDeInit

Function Name	void HAL_TIM_OC_MspDelInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Deinitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

40.3.8.21 HAL_TIM_PWM_MspDelInit

Function Name	void HAL_TIM_PWM_MspDelInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Deinitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

40.3.8.22 HAL_TIM_IC_MspDelInit

Function Name	void HAL_TIM_IC_MspDelInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Deinitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none">• htim : TIM handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

40.3.8.23 HAL_TIM_OnePulse_MspDelInit

Function Name	void HAL_TIM_OnePulse_MspDelInit (<i>TIM_HandleTypeDef</i> * htim)
---------------	--

Function Description	Deinitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.24 HAL_TIM_Encoder_MspDeInit

Function Name	void HAL_TIM_Encoder_MspDeInit (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Deinitializes TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.25 HAL_TIM_Base_Start

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Starts the TIM Base generation.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.26 HAL_TIM_Base_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop (
---------------	--

TIM_HandleTypeDef * htim)

Function Description	Stops the TIM Base generation.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.27 HAL_TIM_Base_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start_IT (<i>TIM_HandleTypeDef * htim)</i>
Function Description	Starts the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.28 HAL_TIM_Base_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (<i>TIM_HandleTypeDef * htim)</i>
Function Description	Stops the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.29 HAL_TIM_Base_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.30 HAL_TIM_Base_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.31 HAL_TIM_OC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channel to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected

- ***TIM_CHANNEL_4*** : TIM Channel 4 selected

Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.32 HAL_TIM_OC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop (<i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.33 HAL_TIM_OC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start_IT (<i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)
Function Description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channel to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

40.3.8.34 HAL_TIM_OC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. Channel : TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected – <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

40.3.8.35 HAL_TIM_OC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. Channel : TIM Channel to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected – <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected pData : The source Buffer address. Length : The length of data to be transferred from memory to TIM peripheral

Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.36 HAL_TIM_OC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected – <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.37 HAL_TIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected – <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.38 HAL_TIM_PWM_Stop

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Stop (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected – <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected – <code>TIM_CHANNEL_3</code> : TIM Channel 3 selected – <code>TIM_CHANNEL_4</code> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.39 HAL_TIM_PWM_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected – <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected – <code>TIM_CHANNEL_3</code> : TIM Channel 3 selected – <code>TIM_CHANNEL_4</code> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.40 HAL_TIM_PWM_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.41 HAL_TIM_PWM_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected • pData : The source Buffer address. • Length : The length of data to be transferred from memory to TIM peripheral
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.42 HAL_TIM_PWM_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.43 HAL_TIM_IC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.44 HAL_TIM_IC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop (<i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)
Function Description	Stops the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.45 HAL_TIM_IC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_IT (<i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.46 HAL_TIM_IC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (<i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)
Function Description	Stops the TIM Input Capture measurement in interrupt mode.

Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.47 HAL_TIM_IC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (<i>TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length</i>)
Function Description	Starts the TIM Input Capture measurement on in DMA mode.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.48 HAL_TIM_IC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (<i>TIM_HandleTypeDef * htim, uint32_t Channel</i>)
Function Description	Stops the TIM Input Capture measurement on in DMA mode.

Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.49 HAL_TIM_OnePulse_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start (<i>TIM_HandleTypeDef * htim, uint32_t OutputChannel</i>)
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – TIM_CHANNEL_1 : TIM Channel 1 selected – TIM_CHANNEL_2 : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.50 HAL_TIM_OnePulse_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (<i>TIM_HandleTypeDef * htim, uint32_t OutputChannel</i>)
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – TIM_CHANNEL_1 : TIM Channel 1 selected – TIM_CHANNEL_2 : TIM Channel 2 selected

Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.51 HAL_TIM_OnePulse_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (<i>TIM_HandleTypeDef</i> * htim , uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • OutputChannel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.52 HAL_TIM_OnePulse_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (<i>TIM_HandleTypeDef</i> * htim , uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. • OutputChannel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.53 HAL_TIM_Encoder_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Start (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Starts the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected – <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.54 HAL_TIM_Encoder_Stop

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Stop (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module. • Channel : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected – <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.55 HAL_TIM_Encoder_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Starts the TIM Encoder Interface in interrupt mode.

Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – TIM_CHANNEL_1 : TIM Channel 1 selected – TIM_CHANNEL_2 : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.56 HAL_TIM_Encoder_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – TIM_CHANNEL_1 : TIM Channel 1 selected – TIM_CHANNEL_2 : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.57 HAL_TIM_Encoder_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – TIM_CHANNEL_1 : TIM Channel 1 selected – TIM_CHANNEL_2 : TIM Channel 2 selected • pData1 : The destination Buffer address for IC1.

	<ul style="list-style-type: none"> pData2 : The destination Buffer address for IC2. Length : The length of data to be transferred from TIM peripheral to memory.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

40.3.8.58 HAL_TIM_Encoder_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

40.3.8.59 HAL_TIM_OC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (<i>TIM_HandleTypeDef</i> * htim, <i>TIM_OC_InitTypeDef</i> * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the <i>TIM_OC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> 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: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected – <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected

Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.60 HAL_TIM_IC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (<i>TIM_HandleTypeDef</i> * htim, <i>TIM_IC_InitTypeDef</i> * sConfig, <i>uint32_t</i> Channel)
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the <i>TIM_IC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> 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: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected – <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

40.3.8.61 HAL_TIM_PWM_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (<i>TIM_HandleTypeDef</i> * htim, <i>TIM_OC_InitTypeDef</i> * sConfig, <i>uint32_t</i> Channel)
Function Description	Initializes the TIM PWM channels according to the specified parameters in the <i>TIM_OC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> 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: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected – <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected

– ***TIM_CHANNEL_4*** : TIM Channel 4 selected

- | | |
|---------------|---|
| Return values | <ul style="list-style-type: none"> • HAL status |
| Notes | <ul style="list-style-type: none"> • None. |

40.3.8.62 HAL_TIM_OnePulse_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (<i>TIM_HandleTypeDef</i> * htim, <i>TIM_OnePulse_InitTypeDef</i> * sConfig, uint32_t OutputChannel, uint32_t InputChannel)
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the <i>TIM_OnePulse_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> 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: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected • InputChannel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected – <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.63 HAL_TIM_DMABurst_WriteStart

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (<i>TIM_HandleTypeDef</i> * 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	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> 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_CCR1*** :
- ***TIM_DMABase_CCR2*** :
- ***TIM_DMABase_CCR3*** :
- ***TIM_DMABase_CCR4*** :
- ***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_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.

40.3.8.64 HAL_TIM_DMABurst_WriteStop

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (<i>TIM_HandleTypeDef</i> * htim, uint32_t BurstRequestSrc)
Function Description	Stops the TIM DMA Burst mode.
Parameters	<ul style="list-style-type: none"> • 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

- None.

40.3.8.65 HAL_TIM_DMABurst_ReadStart

Function Name	<code>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (</code> <code> TIM_HandleTypeDef *htim, uint32_t BurstBaseAddress,</code> <code> uint32_t BurstRequestSrc, uint32_t *BurstBuffer, uint32_t</code> <code> BurstLength)</code>
Function Description	Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <code>TIM_HandleTypeDef</code> 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 one of the following values: <ul style="list-style-type: none"> - <code>TIM_DMABase_CR1</code> : - <code>TIM_DMABase_CR2</code> : - <code>TIM_DMABase_SMCR</code> : - <code>TIM_DMABase_DIER</code> : - <code>TIM_DMABase_SR</code> : - <code>TIM_DMABase_EGR</code> : - <code>TIM_DMABase_CCMR1</code> : - <code>TIM_DMABase_CCMR2</code> : - <code>TIM_DMABase_CCER</code> : - <code>TIM_DMABase_CNT</code> : - <code>TIM_DMABase_PSC</code> : - <code>TIM_DMABase_ARR</code> : - <code>TIM_DMABase_CCR1</code> : - <code>TIM_DMABase_CCR2</code> : - <code>TIM_DMABase_CCR3</code> : - <code>TIM_DMABase_CCR4</code> : - <code>TIM_DMABase_DCR</code> : • BurstRequestSrc : TIM DMA Request sources. This parameters can be one of the following values: <ul style="list-style-type: none"> - <code>TIM_DMA_UPDATE</code> : TIM update Interrupt source - <code>TIM_DMA_CC1</code> : TIM Capture Compare 1 DMA source - <code>TIM_DMA_CC2</code> : TIM Capture Compare 2 DMA source - <code>TIM_DMA_CC3</code> : TIM Capture Compare 3 DMA source - <code>TIM_DMA_CC4</code> : TIM Capture Compare 4 DMA source - <code>TIM_DMA_TRIGGER</code> : TIM Trigger DMA source • BurstBuffer : The Buffer address. • BurstLength : DMA Burst length. This parameter can be one value between <code>TIM_DMABurstLength_1Transfer</code> and <code>TIM_DMABurstLength_18Transfers</code>.
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

40.3.8.66 HAL_TIM_DMABurst_ReadStop

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (<i>TIM_HandleTypeDef</i> * htim, uint32_t BurstRequestSrc)
Function Description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. BurstRequestSrc : TIM DMA Request sources to disable.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

40.3.8.67 HAL_TIM_GenerateEvent

Function Name	HAL_StatusTypeDef HAL_TIM_GenerateEvent (<i>TIM_HandleTypeDef</i> * htim, uint32_t EventSource)
Function Description	Generate a software event.
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module. EventSource : specifies the event source. This parameter can be one of the following values: <ul style="list-style-type: none"> <i>TIM_EventSource_Update</i> : Timer update Event source <i>TIM_EventSource_CC1</i> : Timer Capture Compare 1 Event source <i>TIM_EventSource_CC2</i> : Timer Capture Compare 2 Event source <i>TIM_EventSource_CC3</i> : Timer Capture Compare 3 Event source <i>TIM_EventSource_CC4</i> : Timer Capture Compare 4 Event source <i>TIM_EventSource_Trigger</i> : Timer Trigger Event source
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> TIM6 can only generate an update event.

40.3.8.68 HAL_TIM_ConfigOCrefClear

Function Name	<code>HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (</code> <code> TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef *</code> <code> sClearInputConfig, uint32_t Channel)</code>
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.69 HAL_TIM_ConfigClockSource

Function Name	<code>HAL_StatusTypeDef HAL_TIM_ConfigClockSource (</code> <code> TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef *</code> <code> sClockSourceConfig)</code>
Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.70 HAL_TIM_ConfigTI1Input

Function Name	<code>HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)</code>
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	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.71 HAL_TIM_SlaveConfigSynchronization

Function Name	<code>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)</code>
Function Description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

40.3.8.72 HAL_TIM_ReadCapturedValue

Function Name	<code>uint32_t HAL_TIM_ReadCapturedValue (<i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
Function Description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.. • Channel : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> - <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected - <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected - <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected - <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • Captured value
Notes	<ul style="list-style-type: none"> • None.

40.3.8.73 HAL_TIM_PeriodElapsedCallback

Function Name	<code>void HAL_TIM_PeriodElapsedCallback (<i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None.

Notes

• None.

40.3.8.74 HAL_TIM_OC_DelayElapsedCallback

Function Name	<code>void HAL_TIM_OC_DelayElapsedCallback (<i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Output Compare callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that

contains the configuration information for TIM module.

Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

40.3.8.75 HAL_TIM_IC_CaptureCallback

Function Name	void HAL_TIM_IC_CaptureCallback (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Input Capture callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> htim : TIM IC handle
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

40.3.8.76 HAL_TIM_PWM_PulseFinishedCallback

Function Name	void HAL_TIM_PWM_PulseFinishedCallback (<i>TIM_HandleTypeDef</i> * htim)
Function Description	PWM Pulse finished callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

40.3.8.77 HAL_TIM_TriggerCallback

Function Name	void HAL_TIM_TriggerCallback (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Hall Trigger detection callback in non blocking mode.

Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.78 HAL_TIM_ErrorCallback

Function Name	void HAL_TIM_ErrorCallback (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.79 HAL_TIM_Base_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Return the TIM Base state.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

40.3.8.80 HAL_TIM_OC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (<i>TIM_HandleTypeDef</i> * htim)
---------------	--

Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none"> • htim : TIM Output Compare handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

40.3.8.81 HAL_TIM_PWM_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

40.3.8.82 HAL_TIM_IC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (<i>TIM_HandleTypeDef</i> * htim)
Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

40.3.8.83 HAL_TIM_OnePulse_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (<i>TIM_HandleTypeDef * htim)</i>
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"> • htim : TIM OPM handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

40.3.8.84 HAL_TIM_Encoder_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (<i>TIM_HandleTypeDef * htim)</i>
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"> • htim : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

40.3.8.85 HAL_TIM_IRQHandler

Function Name	void HAL_TIM_IRQHandler (<i>TIM_HandleTypeDef * htim)</i>
Function Description	This function handles TIM interrupts requests.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.86 HAL_TIM_DMAError

Function Name	void HAL_TIM_DMSError (<i>DMA_HandleTypeDef</i> * hdma)
Function Description	TIM DMA error callback.
Parameters	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.87 HAL_TIM_DMADelayPulseCplt

Function Name	void HAL_TIM_DMADelayPulseCplt (<i>DMA_HandleTypeDef</i> * hdma)
Function Description	TIM DMA Delay Pulse complete callback.
Parameters	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.3.8.88 HAL_TIM_DMACaptureCplt

Function Name	void HAL_TIM_DMACaptureCplt (<i>DMA_HandleTypeDef</i> * hdma)
Function Description	TIM DMA Capture complete callback.
Parameters	<ul style="list-style-type: none"> • hdma : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

40.4 TIM Firmware driver defines

40.4.1 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 Tlx clock sources

- #define: **TIM_CLOCKPOLARITY_FALLING
TIM_INPUTCHANNELPOLARITY_FALLING**

Polarity for Tlx 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_CCR1*** (*0x0000000D*)
- #define: ***TIM_DMABase_CCR2*** (*0x0000000E*)
- #define: ***TIM_DMABase_CCR3*** (*0x0000000F*)

- #define: ***TIM_DMABase_CCR4*** (*0x00000010*)
- #define: ***TIM_DMABase_DCR*** (*0x00000012*)
- #define: ***TIM_DMABase_OR*** (*0x00000013*)

TIM_DMABurstLength

- #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_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_EPS_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_Trigger TIM_EGR_TG**

TIM_Exported_Macro

- #define: **CCER_CCxE_MASK ((uint32_t)(TIM_CCER_CC1E | TIM_CCER_CC2E |
TIM_CCER_CC3E | TIM_CCER_CC4E))**

- #define: **__HAL_TIM_PRESCALER (__HANDLE__, __PRESC__) ((__HANDLE__)->Instance->PSC |= (__PRESC__))**

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_TRIGGER (TIM_SR_TIF)***
- #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_TRIGGER (TIM_DIER_TIE)***

TIM_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))***

TIM_Master_Slave_Mode

- #define: ***TIM_MASTERSLAVEMODE_ENABLE ((uint32_t)0x0080)***
- #define: ***TIM_MASTERSLAVEMODE_DISABLE ((uint32_t)0x0000)***

TIM_One_Pulse_Mode

- #define: ***TIM_OPMODE_SINGLE (TIM_CR1_OPM)***
- #define: ***TIM_OPMODE_REPEATITIVE ((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_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)

41 HAL TIM Extension Driver

41.1 TIMEx Firmware driver introduction

41.2 TIMEx Firmware driver registers structures

41.2.1 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the `stm32l0xx_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 [*TIM_Master_Slave_Mode*](#)

41.3 TIMEx Firmware driver API description

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

41.3.1 TIM specific features integration

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for: Input Capture Output Compare PWM generation (Edge and Center-aligned Mode) One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

41.3.2 How to use this driver

1. Enable the TIM interface clock using `__TIMx_CLK_ENABLE()`;
2. 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 operating mode using one of the configuration function of this driver:
 - `HAL_TIMEx_MasterConfigSynchronization()` to configure the peripheral in master mode.
5. Remap the Timer I/O using `HAL_TIMEx_RemapConfig()` API.

41.3.3 Peripheral Control functions

This section provides functions allowing to:

- Configure Master and the Slave synchronization.
- [`HAL_TIMEx_MasterConfigSynchronization\(\)`](#)
- [`HAL_TIMEx_RemapConfig\(\)`](#)

41.3.3.1 `HAL_TIMEx_MasterConfigSynchronization`

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (</code> <code>TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef *</code> <code>sMasterConfig)</code>
Function Description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none"> • <code>htim</code> : TIM handle. • <code>sMasterConfig</code> : pointer to a <code>TIM_MasterConfigTypeDef</code> structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

41.3.3.2 `HAL_TIMEx_RemapConfig`

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_RemapConfig (</code> <code>TIM_HandleTypeDef * htim, uint32_t Remap)</code>
Function Description	Configures the TIM2, TIM21 and TIM22 Remapping input capabilities.

Parameters	
	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - TIM2_ETR_GPIO : TIM2 ETR is connected to GPIO (default) - TIM2_ETR_HSI48 : TIM2 ETR is connected to HSI48 - TIM2_ETR_LSE : TIM2 ETR is connected to LSE - TIM2_ETR_COMP2_OUT : TIM2 ETR is connected to COMP2 output - TIM2_ETR_COMP1_OUT : TIM2 ETR is connected to COMP1 output - TIM2_TI4_GPIO1 : TIM2 TI4 is connected to GPIO1(default) - TIM2_TI4_COMP1 : TIM2 TI4 is connected to COMP1 - TIM2_TI4_COMP2 : TIM2 TI4 is connected to COMP2 - TIM2_TI4_GPIO2 : TIM2 TI4 is connected to GPIO2 - TIM21_ETR_GPIO : TIM21 ETR is connected to GPIO(default) - TIM21_ETR_COMP2_OUT : TIM21 ETR is connected to COMP2 output - TIM21_ETR_COMP1_OUT : TIM21 ETR is connected to COMP1 output - TIM21_ETR_LSE : TIM21 ETR is connected to LSE - TIM21_TI1_MCO : TIM21 TI1 is connected to MCO - TIM21_TI1_RTC_WKUP_IT : TIM21 TI1 is connected to RTC WAKEUP interrupt - TIM21_TI1_HSE_RTC : TIM21 TI1 is connected to HSE_RTC - TIM21_TI1_MSI : TIM21 TI1 is connected to MSI clock - TIM21_TI1_LSE : TIM21 TI1 is connected to LSE - TIM21_TI1_LSI : TIM21 TI1 is connected to LSI - TIM21_TI1_COMP1_OUT : TIM21 TI1 is connected to COMP1_OUT - TIM21_TI1_GPIO : TIM21 TI1 is connected to GPIO(default) - TIM21_TI2_GPIO : TIM21 TI2 is connected to GPIO(default) - TIM21_TI2_COMP2_OUT : TIM21 TI2 is connected to COMP2 output - TIM22_ETR_LSE : TIM22 ETR is connected to LSE - TIM22_ETR_COMP2_OUT : TIM22 ETR is connected to COMP2 output - TIM22_ETR_COMP1_OUT : TIM22 ETR is connected to COMP1 output - TIM22_ETR_GPIO : TIM22 ETR is connected to GPIO(default) - TIM22_TI1_GPIO1 : TIM22 TI1 is connected to GPIO(default) - TIM22_TI1_COMP2_OUT : TIM22 TI1 is connected to COMP2 output - TIM22_TI1_COMP1_OUT : TIM22 TI1 is connected to COMP1 output

– ***TIM22_TI1_GPIO2*** : TIM22 TI1 is connected to GPIO

Return values

- **HAL status**

Notes

- None.

41.4 TIMEx Firmware driver defines

41.4.1 TIMEx

TIMEx

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_Remap



- #define: ***TIM2_ETR_GPIO*** ((*uint32_t*)0xFFFF80000)
- #define: ***TIM2_ETR_HSI48*** ((*uint32_t*)0xFFFF80004)
- #define: ***TIM2_ETR_LSE*** ((*uint32_t*)0xFFFF80005)
- #define: ***TIM2_ETR_COMP2_OUT*** ((*uint32_t*)0xFFFF80006)
- #define: ***TIM2_ETR_COMP1_OUT*** ((*uint32_t*)0xFFFF80007)
- #define: ***TIM2_TI4_GPIO1*** ((*uint32_t*)0xFFE70000)
- #define: ***TIM2_TI4_COMP2*** ((*uint32_t*)0xFFE70008)
- #define: ***TIM2_TI4_COMP1*** ((*uint32_t*)0xFFE70010)
- #define: ***TIM2_TI4_GPIO2*** ((*uint32_t*)0xFFE70018)
- #define: ***TIM21_ETR_GPIO*** ((*uint32_t*)0xFFFF40000)
- #define: ***TIM21_ETR_COMP2_OUT*** ((*uint32_t*)0xFFFF40001)
- #define: ***TIM21_ETR_COMP1_OUT*** ((*uint32_t*)0xFFFF40002)

- #define: ***TIM21_ETR_LSE*** ((*uint32_t*)0xFFFF40003)
- #define: ***TIM21_TI1_MCO*** ((*uint32_t*)0xFFE3001C)
- #define: ***TIM21_TI1_RTC_WKUT_IT*** ((*uint32_t*)0xFFE30004)
- #define: ***TIM21_TI1_HSE_RTC*** ((*uint32_t*)0xFFE30008)
- #define: ***TIM21_TI1_MSI*** ((*uint32_t*)0xFFE3000C)
- #define: ***TIM21_TI1_LSE*** ((*uint32_t*)0xFFE30010)
- #define: ***TIM21_TI1_LSI*** ((*uint32_t*)0xFFE30014)
- #define: ***TIM21_TI1_COMP1_OUT*** ((*uint32_t*)0xFFE30018)
- #define: ***TIM21_TI1_GPIO*** ((*uint32_t*)0xFFE30000)
- #define: ***TIM21_TI2_GPIO*** ((*uint32_t*)0xFFDF0000)
- #define: ***TIM21_TI2_COMP2_OUT*** ((*uint32_t*)0xFFDF0020)
- #define: ***TIM22_ETR_LSE*** ((*uint32_t*)0xFFFFC0000)

-
- #define: ***TIM22_ETR_COMP2_OUT*** ((*uint32_t*)0xFFFFC0001)
 - #define: ***TIM22_ETR_COMP1_OUT*** ((*uint32_t*)0xFFFFC0002)
 - #define: ***TIM22_ETR_GPIO*** ((*uint32_t*)0xFFFFC0003)
 - #define: ***TIM22_TI1_GPIO1*** ((*uint32_t*)0xFFFF70000)
 - #define: ***TIM22_TI1_COMP2_OUT*** ((*uint32_t*)0xFFFF70004)
 - #define: ***TIM22_TI1_COMP1_OUT*** ((*uint32_t*)0xFFFF70008)
 - #define: ***TIM22_TI1_GPIO2*** ((*uint32_t*)0xFFFF7000C)

42 HAL TSC Generic Driver

42.1 TSC Firmware driver introduction

42.2 TSC Firmware driver registers structures

42.2.1 TSC_HandleTypeDef

TSC_HandleTypeDef is defined in the `stm32l0xx_hal_tsc.h`

Data Fields

- *TSC_TypeDef * Instance*
- *TSC_InitTypeDef Init*
- *__IO HAL_TSC_StateTypeDef State*
- *HAL_LockTypeDef Lock*

Field Documentation

- *TSC_TypeDef* TSC_HandleTypeDef::Instance*
 - Register base address
- *TSC_InitTypeDef TSC_HandleTypeDef::Init*
 - Initialization parameters
- *__IO HAL_TSC_StateTypeDef TSC_HandleTypeDef::State*
 - Peripheral state
- *HAL_LockTypeDef TSC_HandleTypeDef::Lock*
 - Lock feature

42.2.2 TSC_InitTypeDef

TSC_InitTypeDef is defined in the `stm32l0xx_hal_tsc.h`

Data Fields

- *uint32_t CTPulseHighLength*
- *uint32_t CTPulseLowLength*
- *uint32_t SpreadSpectrum*
- *uint32_t SpreadSpectrumDeviation*
- *uint32_t SpreadSpectrumPrescaler*
- *uint32_t PulseGeneratorPrescaler*
- *uint32_t MaxCountValue*
- *uint32_t IODefaultMode*
- *uint32_t SynchroPinPolarity*
- *uint32_t AcquisitionMode*
- *uint32_t MaxCountInterrupt*
- *uint32_t ChannelIOs*

-
- *uint32_t ShieldIOs*
 - *uint32_t SamplingIOs*

Field Documentation

- *uint32_t TSC_InitTypeDef::CTPulseHighLength*
 - Charge-transfer high pulse length
- *uint32_t TSC_InitTypeDef::CTPulseLowLength*
 - Charge-transfer low pulse length
- *uint32_t TSC_InitTypeDef::SpreadSpectrum*
 - Spread spectrum activation
- *uint32_t TSC_InitTypeDef::SpreadSpectrumDeviation*
 - Spread spectrum deviation
- *uint32_t TSC_InitTypeDef::SpreadSpectrumPrescaler*
 - Spread spectrum prescaler
- *uint32_t TSC_InitTypeDef::PulseGeneratorPrescaler*
 - Pulse generator prescaler
- *uint32_t TSC_InitTypeDef::MaxCountValue*
 - Max count value
- *uint32_t TSC_InitTypeDef::IODefaultMode*
 - IO default mode
- *uint32_t TSC_InitTypeDef::SynchroPinPolarity*
 - Synchro pin polarity
- *uint32_t TSC_InitTypeDef::AcquisitionMode*
 - Acquisition mode
- *uint32_t TSC_InitTypeDef::MaxCountInterrupt*
 - Max count interrupt activation
- *uint32_t TSC_InitTypeDef::ChannelIOs*
 - Channel IOs mask
- *uint32_t TSC_InitTypeDef::ShieldIOs*
 - Shield IOs mask
- *uint32_t TSC_InitTypeDef::SamplingIOs*
 - Sampling IOs mask

42.2.3 TSC_IOConfigTypeDef

TSC_IOConfigTypeDef is defined in the `stm32l0xx_hal_tsc.h`

Data Fields

- *uint32_t ChannelIOs*
- *uint32_t ShieldIOs*
- *uint32_t SamplingIOs*

Field Documentation

- *uint32_t TSC_IOConfigTypeDef::ChannelIOs*
 - Channel IOs mask

- `uint32_t TSC_IOConfigTypeDef::ShieldIOs`
 - Shield IOs mask
- `uint32_t TSC_IOConfigTypeDef::SamplingIOs`
 - Sampling IOs mask

42.2.4 TSC_TypeDef

`TSC_TypeDef` is defined in the `stm32l052xx.h`

Data Fields

- `__IO uint32_t CR`
- `__IO uint32_t IER`
- `__IO uint32_t ICR`
- `__IO uint32_t ISR`
- `__IO uint32_t IOHCR`
- `uint32_t RESERVED1`
- `__IO uint32_t IOASCR`
- `uint32_t RESERVED2`
- `__IO uint32_t IOSCR`
- `uint32_t RESERVED3`
- `__IO uint32_t IOCCR`
- `uint32_t RESERVED4`
- `__IO uint32_t IOGCSR`
- `__IO uint32_t IOGXCR`

Field Documentation

- `__IO uint32_t TSC_TypeDef::CR`
 - TSC control register, Address offset: 0x00
- `__IO uint32_t TSC_TypeDef::IER`
 - TSC interrupt enable register, Address offset: 0x04
- `__IO uint32_t TSC_TypeDef::ICR`
 - TSC interrupt clear register, Address offset: 0x08
- `__IO uint32_t TSC_TypeDef::ISR`
 - TSC interrupt status register, Address offset: 0x0C
- `__IO uint32_t TSC_TypeDef::IOHCR`
 - TSC I/O hysteresis control register, Address offset: 0x10
- `uint32_t TSC_TypeDef::RESERVED1`
 - Reserved, Address offset: 0x14
- `__IO uint32_t TSC_TypeDef::IOASCR`
 - TSC I/O analog switch control register, Address offset: 0x18
- `uint32_t TSC_TypeDef::RESERVED2`
 - Reserved, Address offset: 0x1C
- `__IO uint32_t TSC_TypeDef::IOSCR`
 - TSC I/O sampling control register, Address offset: 0x20
- `uint32_t TSC_TypeDef::RESERVED3`
 - Reserved, Address offset: 0x24
- `__IO uint32_t TSC_TypeDef::IOCCR`

- TSC I/O channel control register, Address offset: 0x28
- ***uint32_t TSC_TypeDef::RESERVED4***
 - Reserved, Address offset: 0x2C
- ***_IO uint32_t TSC_TypeDef::ILOGCSR***
 - TSC I/O group control status register, Address offset: 0x30
- ***_IO uint32_t TSC_TypeDef::ILOGXCR***
 - TSC I/O group x counter register, Address offset: 0x34-50

42.3 TSC Firmware driver API description

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

42.3.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

42.3.2 How to use this driver

1. Enable the TSC interface clock using `__TSC_CLK_ENABLE()` macro.
2. GPIO pins configuration
 - Enable the clock for the TSC GPIOs using `__GPIOx_CLK_ENABLE()` macro.
 - Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using `HAL_GPIO_Init()` function.
 - Configure the alternate function on all the TSC pins using `HAL_xxxx()` function.
3. Interrupts configuration
 - Configure the NVIC (if the interrupt model is used) using `HAL_xxx()` function.
4. TSC configuration
 - Configure all TSC parameters and used TSC IOs using `HAL_TSC_Init()` function.

Acquisition sequence

- Discharge all IOs using `HAL_TSC_IODischarge()` function.

- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using HAL_TSC_IOConfig() function.
- Launch the acquisition using either HAL_TSC_Start() or HAL_TSC_Start_IT() function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either HAL_TSC_PollForAcquisition() or HAL_TSC_GetState() function or using WFI instruction for example.
- Check the group acquisition status using HAL_TSC_GroupGetStatus() function.
- Read the acquisition value using HAL_TSC_GroupGetValue() function.

42.3.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.
- [**HAL_TSC_Init\(\)**](#)
- [**HAL_TSC_DeInit\(\)**](#)
- [**HAL_TSC_MspInit\(\)**](#)
- [**HAL_TSC_MspDeInit\(\)**](#)

42.3.3.1 HAL_TSC_Init

Function Name	HAL_StatusTypeDef HAL_TSC_Init (TSC_HandleTypeDef *htsc)
Function Description	Initializes the TSC peripheral according to the specified parameters in the TSC_InitTypeDef structure.
Parameters	<ul style="list-style-type: none"> • htsc : TSC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

42.3.3.2 HAL_TSC_DeInit

Function Name	HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef *htsc)
Function Description	Deinitializes the TSC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • htsc : TSC handle
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

42.3.3.3 HAL_TSC_MspInit

Function Name	void HAL_TSC_MspInit (<i>TSC_HandleTypeDef</i> * htsc)
Function Description	Initializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> htsc : pointer to a <i>TSC_HandleTypeDef</i> structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

42.3.3.4 HAL_TSC_MspDelInit

Function Name	void HAL_TSC_MspDelInit (<i>TSC_HandleTypeDef</i> * htsc)
Function Description	Deinitializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> htsc : pointer to a <i>TSC_HandleTypeDef</i> structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

42.4 TSC Firmware driver defines

42.4.1 TSC

TSC

TSC_Exported_Constants

- #define: ***TSC_CTPH_1CYCLE ((uint32_t)(uint32_t) 0 << 28))***

-
- #define: **TSC_CTPH_2CYCLES** ((*uint32_t*)((*uint32_t*) 1 << 28))
 - #define: **TSC_CTPH_3CYCLES** ((*uint32_t*)((*uint32_t*) 2 << 28))
 - #define: **TSC_CTPH_4CYCLES** ((*uint32_t*)((*uint32_t*) 3 << 28))
 - #define: **TSC_CTPH_5CYCLES** ((*uint32_t*)((*uint32_t*) 4 << 28))
 - #define: **TSC_CTPH_6CYCLES** ((*uint32_t*)((*uint32_t*) 5 << 28))
 - #define: **TSC_CTPH_7CYCLES** ((*uint32_t*)((*uint32_t*) 6 << 28))
 - #define: **TSC_CTPH_8CYCLES** ((*uint32_t*)((*uint32_t*) 7 << 28))
 - #define: **TSC_CTPH_9CYCLES** ((*uint32_t*)((*uint32_t*) 8 << 28))
 - #define: **TSC_CTPH_10CYCLES** ((*uint32_t*)((*uint32_t*) 9 << 28))
 - #define: **TSC_CTPH_11CYCLES** ((*uint32_t*)((*uint32_t*) 10 << 28))
 - #define: **TSC_CTPH_12CYCLES** ((*uint32_t*)((*uint32_t*) 11 << 28))
 - #define: **TSC_CTPH_13CYCLES** ((*uint32_t*)((*uint32_t*) 12 << 28))

-
- #define: **TSC_CTPH_14CYCLES** ((*uint32_t*)((*uint32_t*)13 << 28))
 - #define: **TSC_CTPH_15CYCLES** ((*uint32_t*)((*uint32_t*)14 << 28))
 - #define: **TSC_CTPH_16CYCLES** ((*uint32_t*)((*uint32_t*)15 << 28))
 - #define: **TSC_CTPL_1CYCLE** ((*uint32_t*)((*uint32_t*) 0 << 24))
 - #define: **TSC_CTPL_2CYCLES** ((*uint32_t*)((*uint32_t*) 1 << 24))
 - #define: **TSC_CTPL_3CYCLES** ((*uint32_t*)((*uint32_t*) 2 << 24))
 - #define: **TSC_CTPL_4CYCLES** ((*uint32_t*)((*uint32_t*) 3 << 24))
 - #define: **TSC_CTPL_5CYCLES** ((*uint32_t*)((*uint32_t*) 4 << 24))
 - #define: **TSC_CTPL_6CYCLES** ((*uint32_t*)((*uint32_t*) 5 << 24))
 - #define: **TSC_CTPL_7CYCLES** ((*uint32_t*)((*uint32_t*) 6 << 24))
 - #define: **TSC_CTPL_8CYCLES** ((*uint32_t*)((*uint32_t*) 7 << 24))
 - #define: **TSC_CTPL_9CYCLES** ((*uint32_t*)((*uint32_t*) 8 << 24))

- #define: **TSC_CTPL_10CYCLES** ((*uint32_t*)((*uint32_t*) 9 << 24))
- #define: **TSC_CTPL_11CYCLES** ((*uint32_t*)((*uint32_t*)10 << 24))
- #define: **TSC_CTPL_12CYCLES** ((*uint32_t*)((*uint32_t*)11 << 24))
- #define: **TSC_CTPL_13CYCLES** ((*uint32_t*)((*uint32_t*)12 << 24))
- #define: **TSC_CTPL_14CYCLES** ((*uint32_t*)((*uint32_t*)13 << 24))
- #define: **TSC_CTPL_15CYCLES** ((*uint32_t*)((*uint32_t*)14 << 24))
- #define: **TSC_SS_PRESC_DIV1** ((*uint32_t*)0)
- #define: **TSC_SS_PRESC_DIV2** (*TSC_CR_SSPSC*)
- #define: **TSC_PG_PRESC_DIV1** ((*uint32_t*)(0 << 12))
- #define: **TSC_PG_PRESC_DIV2** ((*uint32_t*)(1 << 12))
- #define: **TSC_PG_PRESC_DIV4** ((*uint32_t*)(2 << 12))

-
- #define: **TSC_PG_PRESC_DIV8** ((*uint32_t*)(3 << 12))
 - #define: **TSC_PG_PRESC_DIV16** ((*uint32_t*)(4 << 12))
 - #define: **TSC_PG_PRESC_DIV32** ((*uint32_t*)(5 << 12))
 - #define: **TSC_PG_PRESC_DIV64** ((*uint32_t*)(6 << 12))
 - #define: **TSC_PG_PRESC_DIV128** ((*uint32_t*)(7 << 12))
 - #define: **TSC_MCV_255** ((*uint32_t*)(0 << 5))
 - #define: **TSC_MCV_511** ((*uint32_t*)(1 << 5))
 - #define: **TSC_MCV_1023** ((*uint32_t*)(2 << 5))
 - #define: **TSC_MCV_2047** ((*uint32_t*)(3 << 5))
 - #define: **TSC_MCV_4095** ((*uint32_t*)(4 << 5))
 - #define: **TSC_MCV_8191** ((*uint32_t*)(5 << 5))
 - #define: **TSC_MCV_16383** ((*uint32_t*)(6 << 5))

-
- #define: **TSC_IODEF_OUT_PP_LOW ((uint32_t)0)**
 - #define: **TSC_IODEF_IN_FLOAT (TSC_CR_IODEF)**
 - #define: **TSC_SYNC_POL_FALL ((uint32_t)0)**
 - #define: **TSC_SYNC_POL_RISE_HIGH (TSC_CR_SYNCPOL)**
 - #define: **TSC_ACQ_MODE_NORMAL ((uint32_t)0)**
 - #define: **TSC_ACQ_MODE_SYNCHRO (TSC_CR_SYNCPOL)**
 - #define: **TSC_IOMODE_UNUSED ((uint32_t)0)**
 - #define: **TSC_IOMODE_CHANNEL ((uint32_t)1)**
 - #define: **TSC_IOMODE_SHIELD ((uint32_t)2)**
 - #define: **TSC_IOMODE_SAMPLING ((uint32_t)3)**
 - #define: **TSC_NB_OF_GROUPS (8)**
 - #define: **TSC_GROUP1 ((uint32_t)0x00000001)**

- #define: **TSC_GROUP2** ((*uint32_t*)0x00000002)
- #define: **TSC_GROUP3** ((*uint32_t*)0x00000004)
- #define: **TSC_GROUP4** ((*uint32_t*)0x00000008)
- #define: **TSC_GROUP5** ((*uint32_t*)0x00000010)
- #define: **TSC_GROUP6** ((*uint32_t*)0x00000020)
- #define: **TSC_GROUP7** ((*uint32_t*)0x00000040)
- #define: **TSC_GROUP8** ((*uint32_t*)0x00000080)
- #define: **TSC_ALL_GROUPS** ((*uint32_t*)0x000000FF)
- #define: **TSC_GROUP1_IDX** ((*uint32_t*)0)
- #define: **TSC_GROUP2_IDX** ((*uint32_t*)1)
- #define: **TSC_GROUP3_IDX** ((*uint32_t*)2)
- #define: **TSC_GROUP4_IDX** ((*uint32_t*)3)

-
- #define: **TSC_GROUP5_IDX** ((*uint32_t*)4)
 - #define: **TSC_GROUP6_IDX** ((*uint32_t*)5)
 - #define: **TSC_GROUP7_IDX** ((*uint32_t*)6)
 - #define: **TSC_GROUP8_IDX** ((*uint32_t*)7)
 - #define: **TSC_GROUP1_IO1** ((*uint32_t*)0x00000001)
 - #define: **TSC_GROUP1_IO2** ((*uint32_t*)0x00000002)
 - #define: **TSC_GROUP1_IO3** ((*uint32_t*)0x00000004)
 - #define: **TSC_GROUP1_IO4** ((*uint32_t*)0x00000008)
 - #define: **TSC_GROUP1_ALL_IOS** ((*uint32_t*)0x0000000F)
 - #define: **TSC_GROUP2_IO1** ((*uint32_t*)0x00000010)
 - #define: **TSC_GROUP2_IO2** ((*uint32_t*)0x00000020)
 - #define: **TSC_GROUP2_IO3** ((*uint32_t*)0x00000040)

- #define: **TSC_GROUP2_IO4** ((*uint32_t*)0x00000080)
- #define: **TSC_GROUP2_ALL_IOS** ((*uint32_t*)0x000000F0)
- #define: **TSC_GROUP3_IO1** ((*uint32_t*)0x00000100)
- #define: **TSC_GROUP3_IO2** ((*uint32_t*)0x00000200)
- #define: **TSC_GROUP3_IO3** ((*uint32_t*)0x00000400)
- #define: **TSC_GROUP3_IO4** ((*uint32_t*)0x00000800)
- #define: **TSC_GROUP3_ALL_IOS** ((*uint32_t*)0x00000F00)
- #define: **TSC_GROUP4_IO1** ((*uint32_t*)0x00001000)
- #define: **TSC_GROUP4_IO2** ((*uint32_t*)0x00002000)
- #define: **TSC_GROUP4_IO3** ((*uint32_t*)0x00004000)
- #define: **TSC_GROUP4_IO4** ((*uint32_t*)0x00008000)
- #define: **TSC_GROUP4_ALL_IOS** ((*uint32_t*)0x0000F000)

- #define: **TSC_GROUP5_IO1** ((*uint32_t*)0x00010000)
- #define: **TSC_GROUP5_IO2** ((*uint32_t*)0x00020000)
- #define: **TSC_GROUP5_IO3** ((*uint32_t*)0x00040000)
- #define: **TSC_GROUP5_IO4** ((*uint32_t*)0x00080000)
- #define: **TSC_GROUP5_ALL_IOS** ((*uint32_t*)0x000F0000)
- #define: **TSC_GROUP6_IO1** ((*uint32_t*)0x00100000)
- #define: **TSC_GROUP6_IO2** ((*uint32_t*)0x00200000)
- #define: **TSC_GROUP6_IO3** ((*uint32_t*)0x00400000)
- #define: **TSC_GROUP6_IO4** ((*uint32_t*)0x00800000)
- #define: **TSC_GROUP6_ALL_IOS** ((*uint32_t*)0x00F00000)
- #define: **TSC_GROUP7_IO1** ((*uint32_t*)0x01000000)
- #define: **TSC_GROUP7_IO2** ((*uint32_t*)0x02000000)

- #define: **TSC_GROUP7_IO3** ((*uint32_t*)0x04000000)
- #define: **TSC_GROUP7_IO4** ((*uint32_t*)0x08000000)
- #define: **TSC_GROUP7_ALL_IOS** ((*uint32_t*)0xF0000000)
- #define: **TSC_GROUP8_IO1** ((*uint32_t*)0x10000000)
- #define: **TSC_GROUP8_IO2** ((*uint32_t*)0x20000000)
- #define: **TSC_GROUP8_IO3** ((*uint32_t*)0x40000000)
- #define: **TSC_GROUP8_IO4** ((*uint32_t*)0x80000000)
- #define: **TSC_GROUP8_ALL_IOS** ((*uint32_t*)0xF0000000)
- #define: **TSC_ALL_GROUPS_ALL_IOS** ((*uint32_t*)0xFFFFFFFF)

TSC_flags_definition

- #define: **TSC_FLAG_EOA** ((*uint32_t*)TSC_ISR_EOAF)
- #define: **TSC_FLAG_MCE** ((*uint32_t*)TSC_ISR_MCEF)

TSC_interrupts_definition

- #define: **TSC_IT_EOA** ((*uint32_t*)TSC_IER_EOAIE)

- #define: **TSC_IT_MCE ((uint32_t)TSC_IER_MCEIE)**

43 HAL UART Generic Driver

43.1 UART Firmware driver introduction

43.2 UART Firmware driver registers structures

43.2.1 `UART_HandleTypeDef`

`UART_HandleTypeDef` is defined in the `stm32l0xx_hal_uart.h`

Data Fields

- `USART_TypeDef * Instance`
- `UART_InitTypeDef Init`
- `UART_AdvFeatureInitTypeDef AdvancedInit`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `uint16_t Mask`
- `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`
- `UART_AdvFeatureInitTypeDef UART_HandleTypeDef::AdvancedInit`
- `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`
- `uint16_t UART_HandleTypeDef::Mask`
- `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`

43.2.2 **UART_AdvFeatureInitTypeDef**

UART_AdvFeatureInitTypeDef is defined in the stm32l0xx_hal_uart.h

Data Fields

- **uint32_t AdvFeatureInit**
- **uint32_t TxPinLevellInvert**
- **uint32_t RxPinLevellInvert**
- **uint32_t DataInvert**
- **uint32_t Swap**
- **uint32_t OverrunDisable**
- **uint32_t DMADisableonRxError**
- **uint32_t AutoBaudRateEnable**
- **uint32_t AutoBaudRateMode**
- **uint32_t MSBFirst**

Field Documentation

- **uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit**
 - Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of [UART_Advanced_Features_Initialization_Type](#)
- **uint32_t UART_AdvFeatureInitTypeDef::TxPinLevellInvert**
 - Specifies whether the TX pin active level is inverted. This parameter can be a value of [UART_Tx_Inv](#)
- **uint32_t UART_AdvFeatureInitTypeDef::RxPinLevellInvert**
 - Specifies whether the RX pin active level is inverted. This parameter can be a value of [UART_Rx_Inv](#)
- **uint32_t UART_AdvFeatureInitTypeDef::DataInvert**
 - Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of [UART_Data_Inv](#)
- **uint32_t UART_AdvFeatureInitTypeDef::Swap**
 - Specifies whether TX and RX pins are swapped. This parameter can be a value of [UART_Rx_Tx_Swap](#)
- **uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable**
 - Specifies whether the reception overrun detection is disabled. This parameter can be a value of [UART_Overrun_Disable](#)
- **uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError**
 - Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of [UART_DMA_Disable_on_Rx_Error](#)
- **uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable**
 - Specifies whether auto Baud rate detection is enabled. This parameter can be a value of [UART_AutoBaudRate_Enable](#)
- **uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode**
 - If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of [UARTE_AutoBaud_Rate_Mode](#)
- **uint32_t UART_AdvFeatureInitTypeDef::MSBFirst**
 - Specifies whether MSB is sent first on UART line. This parameter can be a value of [UART_MSB_First](#)

43.2.3 **UART_InitTypeDef**

UART_InitTypeDef is defined in the stm32l0xx_hal_uart.h

Data Fields

- **uint32_t BaudRate**
- **uint32_t WordLength**
- **uint32_t StopBits**
- **uint32_t Parity**
- **uint32_t Mode**
- **uint32_t HwFlowCtl**
- **uint32_t OverSampling**
- **uint32_t OneBitSampling**

Field Documentation

- **uint32_t UART_InitTypeDef::BaudRate**
 - This member configures the UART communication baud rate. The baud rate register is computed using the following formula: If oversampling is 16 or in LIN mode, Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))If oversampling is 8, Baud Rate Register[15:4] = ((2 * PCLKx) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * PCLKx) / ((huart->Init.BaudRate)))[3:0]) >> 1
- **uint32_t UART_InitTypeDef::WordLength**
 - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UARTEx_Word_Length](#)
- **uint32_t UART_InitTypeDef::StopBits**
 - Specifies the number of stop bits transmitted. This parameter can be a value of [UART_Stop_Bits](#)
- **uint32_t UART_InitTypeDef::Parity**
 - Specifies the parity mode. This parameter can be a value of [UART_Parity](#)
- **uint32_t UART_InitTypeDef::Mode**
 - Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [UART_Mode](#)
- **uint32_t UART_InitTypeDef::HwFlowCtl**
 - Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [UART_Hardware_Flow_Control](#)
- **uint32_t UART_InitTypeDef::OverSampling**
 - Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [UART_Over_Sampling](#)
- **uint32_t UART_InitTypeDef::OneBitSampling**
 - Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [UART_OneBit_Sampling](#)

43.2.4 **USART_TypeDef**

USART_TypeDef is defined in the stm32l051xx.h

Data Fields

- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t CR3`
- `__IO uint32_t BRR`
- `__IO uint16_t GTPR`
- `uint16_t RESERVED2`
- `__IO uint32_t RTOR`
- `__IO uint16_t RQR`
- `uint16_t RESERVED3`
- `__IO uint32_t ISR`
- `__IO uint32_t ICR`
- `__IO uint16_t RDR`
- `uint16_t RESERVED4`
- `__IO uint16_t TDR`
- `uint16_t RESERVED5`

Field Documentation

- `__IO uint32_t USART_TypeDef::CR1`
 - USART Control register 1, Address offset: 0x00
- `__IO uint32_t USART_TypeDef::CR2`
 - USART Control register 2, Address offset: 0x04
- `__IO uint32_t USART_TypeDef::CR3`
 - USART Control register 3, Address offset: 0x08
- `__IO uint32_t USART_TypeDef::BRR`
 - USART Baud rate register, Address offset: 0x0C
- `__IO uint16_t USART_TypeDef::GTPR`
 - USART Guard time and prescaler register, Address offset: 0x10
- `uint16_t USART_TypeDef::RESERVED2`
 - Reserved, 0x12
- `__IO uint32_t USART_TypeDef::RTOR`
 - USART Receiver Time Out register, Address offset: 0x14
- `__IO uint16_t USART_TypeDef::RQR`
 - USART Request register, Address offset: 0x18
- `uint16_t USART_TypeDef::RESERVED3`
 - Reserved, 0x1A
- `__IO uint32_t USART_TypeDef::ISR`
 - USART Interrupt and status register, Address offset: 0x1C
- `__IO uint32_t USART_TypeDef::ICR`
 - USART Interrupt flag Clear register, Address offset: 0x20
- `__IO uint16_t USART_TypeDef::RDR`
 - USART Receive Data register, Address offset: 0x24
- `uint16_t USART_TypeDef::RESERVED4`
 - Reserved, 0x26
- `__IO uint16_t USART_TypeDef::TDR`
 - USART Transmit Data register, Address offset: 0x28
- `uint16_t USART_TypeDef::RESERVED5`
 - Reserved, 0x2A

43.3 UART Firmware driver API description

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

43.3.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits).
 - Hardware flow control
 - Receiver/transmitter modes
 - Over Sampling Method
 - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
 - TX and/or RX pin level inversion
 - data logical level inversion
 - RX and TX pins swap
 - RX overrun detection disabling
 - DMA disabling on RX error
 - MSB first on communication line
 - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init() and HAL_MultiProcessorEx_Init() API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

- [*HAL_UART_Init\(\)*](#)
- [*HAL_HalfDuplex_Init\(\)*](#)
- [*HAL_LIN_Init\(\)*](#)
- [*HAL_MultiProcessor_Init\(\)*](#)
- [*HAL_UART_DeInit\(\)*](#)
- [*HAL_UART_MspInit\(\)*](#)
- [*HAL_UART_MspDeInit\(\)*](#)

43.3.2 IO operation functions

- [*HAL_UART_Transmit\(\)*](#)
- [*HAL_UART_Receive\(\)*](#)
- [*HAL_UART_Transmit_IT\(\)*](#)
- [*HAL_UART_Receive_IT\(\)*](#)
- [*HAL_UART_Transmit_DMA\(\)*](#)
- [*HAL_UART_Receive_DMA\(\)*](#)
- [*HAL_UART_DMADelete\(\)*](#)
- [*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\(\)`](#)
- [`HAL_UART_WakeupCallback\(\)`](#)

43.3.3 Peripheral Control functions

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

- `HAL_MultiProcessor_EnableMuteMode()` API enables mute mode
- `HAL_MultiProcessor_DisableMuteMode()` API disables mute mode
- `HAL_MultiProcessor_EnterMuteMode()` API enters mute mode
- `HAL_HalfDuplex_EnableTransmitter()` API enables the transmitter
- `HAL_HalfDuplex_EnableReceiver()` API enables the receiver
- `HAL_UART_GetState()` API is helpful to check in run-time the state of the UART peripheral
- `HAL_UART_GetError()` API is helpful to check in run-time the error state of the UART peripheral
- [`HAL_MultiProcessor_EnableMuteMode\(\)`](#)
- [`HAL_MultiProcessor_DisableMuteMode\(\)`](#)
- [`HAL_MultiProcessor_EnterMuteMode\(\)`](#)
- [`HAL_HalfDuplex_EnableTransmitter\(\)`](#)
- [`HAL_HalfDuplex_EnableReceiver\(\)`](#)
- [`HAL_LIN_SendBreak\(\)`](#)
- [`HAL_UART_GetState\(\)`](#)
- [`HAL_UART_GetError\(\)`](#)

43.3.3.1 `HAL_UART_Init`

Function Name	<code>HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)</code>
Function Description	Initializes the UART mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • <code>huart</code> : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.2 `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 creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.3 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 creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart : uart handle • BreakDetectLength : specifies the LIN break detection length. This parameter can be one of the following values: <ul style="list-style-type: none"> – UART_LINBREAKDETECTLENGTH_10B : 10-bit break detection – UART_LINBREAKDETECTLENGTH_11B : 11-bit break detection
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.4 HAL_MultiProcessor_Init

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)
Function Description	Initializes the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • huart : UART handle

	<ul style="list-style-type: none"> • Address : UART node address (4-, 6-, 7- or 8-bit long) • WakeUpMethod : specifies the UART wakeup method. This parameter can be one of the following values: <ul style="list-style-type: none"> – <i>UART_WAKEUPMETHOD_IDLELINE</i> : WakeUp by an idle line detection – <i>UART_WAKEUPMETHOD_ADDRESSMARK</i> : WakeUp by an address mark
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function. • If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

43.3.3.5 HAL_UART_DeInit

Function Name	HAL_StatusTypeDef HAL_UART_DeInit (<i>UART_HandleTypeDef</i> * huart)
Function Description	DeInitializes the UART peripheral.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.6 HAL_UART_MspInit

Function Name	void HAL_UART_MspInit (<i>UART_HandleTypeDef</i> * huart)
Function Description	UART MSP Init.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

43.3.3.7 HAL_UART_MspDeInit

Function Name	void HAL_UART_MspDeInit (<i>UART_HandleTypeDef</i> * huart)
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

43.3.3.8 HAL_UART_Transmit

Function Name	HAL_StatusTypeDef HAL_UART_Transmit (<i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle • pData : pointer to data buffer • Size : amount of data to be sent • Timeout : : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.9 HAL_UART_Receive

Function Name	HAL_StatusTypeDef HAL_UART_Receive (<i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle • pData : pointer to data buffer

-
- | | |
|---|--|
| Return values | <ul style="list-style-type: none"> • Size : amount of data to be received • Timeout : Timeout duration |
| Notes | <ul style="list-style-type: none"> • HAL status |
| <ul style="list-style-type: none"> • None. | |

43.3.3.10 HAL_UART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_IT (<i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle • pData : pointer to data buffer • Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.11 HAL_UART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_UART_Receive_IT (<i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle • pData : pointer to data buffer • Size : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.12 HAL_UART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle • pData : pointer to data buffer • Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.13 HAL_UART_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle • pData : pointer to data buffer • Size : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the UART parity is enabled (PCE = 1) the data received contain the parity bit.

43.3.3.14 HAL_UART_DMAPause

Function Name	HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • huart : UART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

43.3.3.15 HAL_UART_DMAResume

Function Name	HAL_StatusTypeDef HAL_UART_DMAResume (<i>UART_HandleTypeDef * huart</i>)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none">• huart : UART handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

43.3.3.16 HAL_UART_DMAStop

Function Name	HAL_StatusTypeDef HAL_UART_DMAStop (<i>UART_HandleTypeDef * huart</i>)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none">• huart : UART handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

43.3.3.17 HAL_UART_IRQHandler

Function Name	void HAL_UART_IRQHandler (<i>UART_HandleTypeDef * huart</i>)
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none">• huart : uart handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

43.3.3.18 HAL_UART_TxCpltCallback

Function Name	void HAL_UART_TxCpltCallback (<i>UART_HandleTypeDef</i> * huart)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

43.3.3.19 HAL_UART_TxHalfCpltCallback

Function Name	void HAL_UART_TxHalfCpltCallback (<i>UART_HandleTypeDef</i> * huart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • huart : UART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

43.3.3.20 HAL_UART_RxCpltCallback

Function Name	void HAL_UART_RxCpltCallback (<i>UART_HandleTypeDef</i> * huart)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

43.3.3.21 HAL_UART_RxHalfCpltCallback

Function Name	void HAL_UART_RxHalfCpltCallback (<i>UART_HandleTypeDef</i> * huart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• huart : UART handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

43.3.3.22 HAL_UART_ErrorCallback

Function Name	void HAL_UART_ErrorCallback (<i>UART_HandleTypeDef</i> * huart)
Function Description	UART error callbacks.
Parameters	<ul style="list-style-type: none">• huart : uart handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

43.3.3.23 HAL_UART_WakeupCallback

Function Name	void HAL_UART_WakeupCallback (<i>UART_HandleTypeDef</i> * huart)
Function Description	UART wakeup from Stop mode callback.
Parameters	<ul style="list-style-type: none">• huart : uart handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

43.3.3.24 HAL_MultiProcessor_EnableMuteMode

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (<i>UART_HandleTypeDef * huart</i>)
Function Description	Enable UART in mute mode (doesn't mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called)
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.25 HAL_MultiProcessor_DisableMuteMode

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (<i>UART_HandleTypeDef * huart</i>)
Function Description	Disable UART mute mode (doesn't mean it actually wakes up the software, as it may not have been in mute mode at this very moment).
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.26 HAL_MultiProcessor_EnterMuteMode

Function Name	void HAL_MultiProcessor_EnterMuteMode (<i>UART_HandleTypeDef * huart</i>)
Function Description	Enter UART mute mode (means UART actually enters mute mode).
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

43.3.3.27 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	<ul style="list-style-type: none">• huart : UART handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

43.3.3.28 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	<ul style="list-style-type: none">• huart : UART handle
Return values	<ul style="list-style-type: none">• HAL status
Notes	<ul style="list-style-type: none">• None.

43.3.3.29 HAL_LIN_SendBreak

Function Name	HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)
Function Description	Transmits break characters.
Parameters	<ul style="list-style-type: none">• huart : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• HAL status

Notes	<ul style="list-style-type: none"> None.
-------	---

43.3.3.30 HAL_UART_GetState

Function Name	<code>HAL_UART_StateTypeDef HAL_UART_GetState (</code> <code>UART_HandleTypeDef * huart)</code>
Function Description	return the UART state
Parameters	<ul style="list-style-type: none"> huart : uart handle
Return values	<ul style="list-style-type: none"> HAL state
Notes	<ul style="list-style-type: none"> None.

43.3.3.31 HAL_UART_GetError

Function Name	<code>uint32_t HAL_UART_GetError (</code> <code>UART_HandleTypeDef * huart)</code>
Function Description	Return the UART error code.
Parameters	<ul style="list-style-type: none"> huart : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none"> UART Error Code
Notes	<ul style="list-style-type: none"> None.

43.4 UART Firmware driver defines

43.4.1 UART

UART

`UART_Advanced_Features_Initialization_Type`

- #define: `UART_ADVFEATURE_NO_INIT ((uint32_t)0x00000000)`

- #define: **UART_ADVFEATURE_TXINVERT_INIT** ((*uint32_t*)0x00000001)
- #define: **UART_ADVFEATURE_RXINVERT_INIT** ((*uint32_t*)0x00000002)
- #define: **UART_ADVFEATURE_DATAINVERT_INIT** ((*uint32_t*)0x00000004)
- #define: **UART_ADVFEATURE_SWAP_INIT** ((*uint32_t*)0x00000008)
- #define: **UART_ADVFEATURE_RXOVERRUNDISABLE_INIT** ((*uint32_t*)0x00000010)
- #define: **UART_ADVFEATURE_DMADISABLEONERROR_INIT** ((*uint32_t*)0x00000020)
- #define: **UART_ADVFEATURE_AUTOBAUDRATE_INIT** ((*uint32_t*)0x00000040)
- #define: **UART_ADVFEATURE_MSBFIRST_INIT** ((*uint32_t*)0x00000080)

UART_AutoBaudRate_Enable

- #define: **UART_ADVFEATURE_AUTOBAUDRATE_DISABLE** ((*uint32_t*)0x00000000)
- #define: **UART_ADVFEATURE_AUTOBAUDRATE_ENABLE** ((*uint32_t*)**USART_CR2_ABREN**)

UART_CR1_DEAT_ADDRESS_LSBPOS

- #define: **UART_CR1_DEAT_ADDRESS_LSB_POS** ((*uint32_t*) 21)

UART_CR1_DEDT_ADDRESS_LSBPOS

- #define: ***UART_CR1_DEDT_ADDRESS_LSB_POS ((uint32_t) 16)***

UART_CR2_ADDRESS_LSBPOS

- #define: ***UART_CR2_ADDRESS_LSB_POS ((uint32_t) 24)***

UART_Data_Inv

- #define: ***UART_ADVFEATURE_DATAINV_DISABLE ((uint32_t)0x00000000)***
- #define: ***UART_ADVFEATURE_DATAINV_ENABLE ((uint32_t)USART_CR2_DATAINV)***

UART_DMA_Disable_on_Rx_Error

- #define: ***UART_ADVFEATURE_DMA_ENABLEONRXERROR ((uint32_t)0x00000000)***
- #define: ***UART_ADVFEATURE_DMA_DISABLEONRXERROR ((uint32_t)USART_CR3_DDRE)***

UART_DMA_Rx

- #define: ***UART_DMA_RX_DISABLE ((uint32_t)0x0000)***
- #define: ***UART_DMA_RX_ENABLE ((uint32_t)USART_CR3_DMAR)***

UART_DMA_Tx

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

-
- #define: **UART_DMA_TX_ENABLE** ((*uint32_t*)USART_CR3_DMAT)

UART_DriverEnable_Polarity

- #define: **UART_DE_POLARITY_HIGH** ((*uint32_t*)0x00000000)
- #define: **UART_DE_POLARITY_LOW** ((*uint32_t*)USART_CR3_DEP)

UART_Flags

- #define: **UART_FLAG_RXACK** ((*uint32_t*)0x00400000)
- #define: **UART_FLAG_TEACK** ((*uint32_t*)0x00200000)
- #define: **UART_FLAG_WUF** ((*uint32_t*)0x00100000)
- #define: **UART_FLAG_RWU** ((*uint32_t*)0x00080000)
- #define: **UART_FLAG_SBKF** ((*uint32_t*)0x00040000)
- #define: **UART_FLAG_CMF** ((*uint32_t*)0x00020000)
- #define: **UART_FLAG_BUSY** ((*uint32_t*)0x00010000)
- #define: **UART_FLAG_ABRF** ((*uint32_t*)0x00008000)
- #define: **UART_FLAG_ABRE** ((*uint32_t*)0x00004000)

- #define: ***UART_FLAG_EOBF*** ((*uint32_t*)0x00001000)
- #define: ***UART_FLAG_RTOF*** ((*uint32_t*)0x00000800)
- #define: ***UART_FLAG_CTS*** ((*uint32_t*)0x00000400)
- #define: ***UART_FLAG_CTSIF*** ((*uint32_t*)0x00000200)
- #define: ***UART_FLAG_LBDF*** ((*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_Half_Duplex_Selection

- #define: **UART_HALF_DUPLEX_DISABLE** ((*uint32_t*)0x0000)
- #define: **UART_HALF_DUPLEX_ENABLE** ((*uint32_t*)USART_CR3_HDSEL)

UART_Hardware_Flow_Control

- #define: **UART_HWCONTROL_NONE** ((*uint32_t*)0x0000)
- #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_Interruption_Mask

- #define: **UART_IT_MASK** ((*uint32_t*)0x001F)

UART Interrupt definition

- #define: **UART_IT_PE** ((*uint32_t*)0x0028)
- #define: **UART_IT_TXE** ((*uint32_t*)0x0727)

- #define: **UART_IT_TC** ((*uint32_t*)0x0626)
 - #define: **UART_IT_RXNE** ((*uint32_t*)0x0525)
 - #define: **UART_IT_IDLE** ((*uint32_t*)0x0424)
 - #define: **UART_IT_LBD** ((*uint32_t*)0x0846)
 - #define: **UART_IT_CTS** ((*uint32_t*)0x096A)
 - #define: **UART_IT_CM** ((*uint32_t*)0x142E)
 - #define: **UART_IT_WUF** ((*uint32_t*)0x1476)
 - #define: **UART_IT_ERR** ((*uint32_t*)0x0060)
YYYYYY : Interrupt source position in the XX register (2bits)
01: CR1 register
10: CR2 register
11: CR3 register
 - #define: **UART_IT_ORE** ((*uint32_t*)0x0300)
ZZZZ : Flag position in the ISR register (4bits)
 - #define: **UART_IT_NE** ((*uint32_t*)0x0200)
 - #define: **UART_IT_FE** ((*uint32_t*)0x0100)

UART IT CLEAR Flags

- #define: ***UART_CLEAR_PEF USART_ICR_PECF***
Parity Error Clear Flag

- #define: **UART_CLEAR_FEF USART_ICR_FECF**
Framing Error Clear Flag
- #define: **UART_CLEAR_NEF USART_ICR_NCF**
Noise detected Clear Flag
- #define: **UART_CLEAR_OREF USART_ICR_ORECF**
OverRun Error Clear Flag
- #define: **UART_CLEAR_IDLEF USART_ICR_IDLECF**
IDLE line detected Clear Flag
- #define: **UART_CLEAR_TCF USART_ICR_TCCF**
Transmission Complete Clear Flag
- #define: **UART_CLEAR_LBDF USART_ICR_LBDCF**
LIN Break Detection Clear Flag
- #define: **UART_CLEAR_CTSF USART_ICR_CTSCF**
CTS Interrupt Clear Flag
- #define: **UART_CLEAR_RTOF USART_ICR_RTOCF**
Receiver Time Out Clear Flag
- #define: **UART_CLEAR_EOBF USART_ICR_EOBCF**
End Of Block Clear Flag
- #define: **UART_CLEAR_CMF USART_ICR_CMCF**
Character Match Clear Flag
- #define: **UART_CLEAR_WUF USART_ICR_WUCF**
Wake Up from stop mode Clear Flag

UART_LIN

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

- #define: **UART_LIN_ENABLE ((uint32_t)USART_CR2_LINEN)**

UART_LIN_Break_Detection

- #define: **UART_LINBREAKDETECTLENGTH_10B ((uint32_t)0x00000000)**
- #define: **UART_LINBREAKDETECTLENGTH_11B ((uint32_t)USART_CR2_LBDL)**

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_MSB_First

- #define: **UART_ADVFEATURE_MSBFIRST_DISABLE ((uint32_t)0x00000000)**
- #define: **UART_ADVFEATURE_MSBFIRST_ENABLE ((uint32_t)USART_CR2_MSBFIRST)**

UART_Mute_Mode

- #define: **UART_ADVFEATURE_MUTEMODE_DISABLE ((uint32_t)0x00000000)**
- #define: **UART_ADVFEATURE_MUTEMODE_ENABLE ((uint32_t)USART_CR1_MME)**

UART_OneBit_Sampling

- #define: ***UART_ONEBIT_SAMPLING_DISABLED*** ((*uint32_t*)0x0000)
- #define: ***UART_ONEBIT_SAMPLING_ENABLED*** ((*uint32_t*)***USART_CR3_ONEBIT***)

UART_One_Bit

- #define: ***UART_ONE_BIT_SAMPLE_DISABLED*** ((*uint32_t*)0x00000000)
- #define: ***UART_ONE_BIT_SAMPLE_ENABLED*** ((*uint32_t*)***USART_CR3_ONEBIT***)

UART_OVERRUN_Disable

- #define: ***UART_ADVFEATURE_OVERRUN_ENABLE*** ((*uint32_t*)0x00000000)
- #define: ***UART_ADVFEATURE_OVERRUN_DISABLE*** ((*uint32_t*)***USART_CR3_OVRDIS***)

UART_Over_Sampling

- #define: ***UART_OVERSAMPLING_16*** ((*uint32_t*)0x0000)
- #define: ***UART_OVERSAMPLING_8*** ((*uint32_t*)***USART_CR1_OVER8***)

UART_Parity

- #define: ***UART_PARITY_NONE*** ((*uint32_t*)0x0000)
- #define: ***UART_PARITY_EVEN*** ((*uint32_t*)***USART_CR1_PCE***)

-
- #define: ***UART_PARITY_ODD*** ((*uint32_t*)(*USART_CR1_PCE* | *USART_CR1_PS*))

UART_Receiver_TimeOut

- #define: ***UART_RECEIVER_TIMEOUT_DISABLE*** ((*uint32_t*)0x00000000)
- #define: ***UART_RECEIVER_TIMEOUT_ENABLE*** ((*uint32_t*)*USART_CR2_RTOEN*)

UART_Request_Parameters

- #define: ***UART_AUTOBAUD_REQUEST*** ((*uint32_t*)*USART_RQR_ABRRQ*)
Auto-Baud Rate Request
- #define: ***UART_SENDBREAK_REQUEST*** ((*uint32_t*)*USART_RQR_SBKRQ*)
Send Break Request
- #define: ***UART_MUTE_MODE_REQUEST*** ((*uint32_t*)*USART_RQR_MMRQ*)
Mute Mode Request
- #define: ***UART_RXDATA_FLUSH_REQUEST*** ((*uint32_t*)*USART_RQR_RXFRQ*)
Receive Data flush Request
- #define: ***UART_TXDATA_FLUSH_REQUEST*** ((*uint32_t*)*USART_RQR_TXFRQ*)
Transmit data flush Request

UART_Rx_Inv

- #define: ***UART_ADVFEATURE_RXINV_DISABLE*** ((*uint32_t*)0x00000000)
- #define: ***UART_ADVFEATURE_RXINV_ENABLE*** ((*uint32_t*)*USART_CR2_RXINV*)

UART_Rx_Tx_Swap

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

- #define: **UART_ADVFEATURE_SWAP_ENABLE** ((*uint32_t*)USART_CR2_SWAP)

UART_State

- #define: **UART_STATE_DISABLE** ((*uint32_t*)0x0000)
- #define: **UART_STATE_ENABLE** ((*uint32_t*)USART_CR1_UE)

UART_Stop_Bits

- #define: **UART_STOPBITS_1** ((*uint32_t*)0x0000)
- #define: **UART_STOPBITS_2** ((*uint32_t*)USART_CR2_STOP_1)

UART_Stop_Mode_Enable

- #define: **UART_ADVFEATURE_STOPMODE_DISABLE** ((*uint32_t*)0x00000000)
- #define: **UART_ADVFEATURE_STOPMODE_ENABLE** ((*uint32_t*)USART_CR1_UESM)

UART_Tx_Inv

- #define: **UART_ADVFEATURE_TXINV_DISABLE** ((*uint32_t*)0x00000000)
- #define: **UART_ADVFEATURE_TXINV_ENABLE** ((*uint32_t*)USART_CR2_TXINV)

UART_WakeUp_from_Stop_Selection

- #define: **UART_WAKEUP_ON_ADDRESS** ((*uint32_t*)0x0000)

-
- #define: ***UART_WAKEUP_ON_STARTBIT*** ((*uint32_t*)***USART_CR3_WUS_1***)
 - #define: ***UART_WAKEUP_ON_READDATA_NONEMPTY*** ((*uint32_t*)***USART_CR3_WUS***)

44 HAL UART Extension Driver

44.1 UARTEEx Firmware driver introduction

44.2 UARTEEx Firmware driver registers structures

44.2.1 `UART_WakeUpTypeDef`

`UART_WakeUpTypeDef` is defined in the `stm32l0xx_hal_uart_ex.h`

Data Fields

- `uint32_t WakeUpEvent`
- `uint16_t AddressLength`
- `uint8_t Address`

Field Documentation

- `uint32_t UART_WakeUpTypeDef::WakeUpEvent`
 - Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of [`UART_WakeUp_from_Stop_Selection`](#).
- `uint16_t UART_WakeUpTypeDef::AddressLength`
 - Specifies whether the address is 4 or 7-bit long. This parameter can be a value of [`UARTEEx_WakeUp_Address_Length`](#)
- `uint8_t UART_WakeUpTypeDef::Address`
 - UART/USART node address (7-bit long max)

44.3 UARTEEx Firmware driver API description

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

44.3.1 Initialization and Configuration functions

The `HAL_RS485Ex_Init()` API follows respectively the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

- [`HAL_RS485Ex_Init\(\)`](#)

44.3.2 Peripheral Control functions

This section provides functions allowing to:

- `UART_AdvFeatureConfig()` API optionally configures the UART advanced features

- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_EnableClockStopMode() API enables the UART HSI clock during stop mode
- HAL_UARTEx_DisableClockStopMode() API disables the above functionality
- UART_Wakeup_AddressConfig() API configures the wake-up from stop mode parameters
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_EnableClockStopMode()***
- ***HAL_UARTEx_DisableStopMode()***
- ***HAL_UARTEx_DisableClockStopMode()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_MultiProcessorEx_AddressLength_Set()***

44.3.2.1 HAL_RS485Ex_Init

Function Name	HAL_StatusTypeDef HAL_RS485Ex_Init (<i>UART_HandleTypeDef</i> * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)
Function Description	Initializes the RS485 Driver enable feature according to the specified parameters in the <i>UART_InitTypeDef</i> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart : uart handle • Polarity : select the driver enable polarity This parameter can be one of the following: <ul style="list-style-type: none"> – <i>UART_DE_POLARITY_HIGH</i> : DE signal is active high – <i>UART_DE_POLARITY_LOW</i> : DE signal is active low • AssertionTime : Driver Enable assertion time 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate) • DeassertionTime : Driver Enable deassertion time 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

44.3.2.2 HAL_UARTEx_EnableStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (<i>UART_HandleTypeDef * huart</i>)
Function Description	Enable UART Stop Mode The UART is able to wake up the MCU from Stop mode as long as UART clock is HSI or LSE.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

44.3.2.3 HAL_UARTEx_EnableClockStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_EnableClockStopMode (<i>UART_HandleTypeDef * huart</i>)
Function Description	Enable UART Clock in Stop Mode The UART keeps the Clock ON during Stop mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

44.3.2.4 HAL_UARTEx_DisableStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (<i>UART_HandleTypeDef * huart</i>)
Function Description	Disable UART Stop Mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

44.3.2.5 HAL_UARTEx_DisableClockStopMode

Function Name	<code>HAL_StatusTypeDef HAL_UARTEx_DisableClockStopMode (UART_HandleTypeDef * huart)</code>
Function Description	Disable UART Clock in Stop Mode.
Parameters	<ul style="list-style-type: none"> • huart : uart handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

44.3.2.6 HAL_UARTEx_StopModeWakeUpSourceConfig

Function Name	<code>HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)</code>
Function Description	Set Wakeup from Stop mode interrupt flag selection.
Parameters	<ul style="list-style-type: none"> • huart : uart handle, • WakeUpSelection : address match, Start Bit detection or RXNE bit status. This parameter can be one of the following values: <ul style="list-style-type: none"> - UART_WAKEUP_ON_ADDRESS : - UART_WAKEUP_ON_STARTBIT : - UART_WAKEUP_ON_RXNE :
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

44.3.2.7 HAL_MultiProcessorEx_AddressLength_Set

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)</code>
---------------	--

Function Description	By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection.
Notes	<ul style="list-style-type: none"> None.

44.4 UARTEEx Firmware driver defines

44.4.1 UARTEEx

UARTEEx

UARTEEx_AutoBaud_Rate_Mode

- #define: **UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT**
((uint32_t)0x0000)
- #define: **UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE**
((uint32_t)USART_CR2_ABRMODE_0)
- #define: **UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFFRAME**
((uint32_t)USART_CR2_ABRMODE_1)
- #define: **UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME**
((uint32_t)USART_CR2_ABRMODE)

UARTEEx_WakeUp_Address_Length

- #define: **UART_ADDRESS_DETECT_4B** ((uint32_t)0x00000000)
- #define: **UART_ADDRESS_DETECT_7B** ((uint32_t)USART_CR2_ADDM7)

UARTEEx_WakeUp_Methods

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

- #define: ***UART_WAKEUPMETHOD_ADDRESSMARK***
((*uint32_t*)***USART_CR1_WAKE***)

UARTEx_Word_Length

- #define: ***UART_WORDLENGTH_7B*** ((*uint32_t*)***USART_CR1_M_1***)
- #define: ***UART_WORDLENGTH_8B*** ((*uint32_t*)***0x0000***)
- #define: ***UART_WORDLENGTH_9B*** ((*uint32_t*)***USART_CR1_M_0***)

45 HAL USART Generic Driver

45.1 USART Firmware driver introduction

45.2 USART Firmware driver registers structures

45.2.1 USART_HandleTypeDef

USART_HandleTypeDef is defined in the `stm32l0xx_hal_usart.h`

Data Fields

- *USART_TypeDef * Instance*
- *USART_InitTypeDef Init*
- *uint8_t * pTxBuffPtr*
- *uint16_t TxXferSize*
- *__IO uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *uint16_t Mask*
- *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 registers base address
- ***USART_InitTypeDef USART_HandleTypeDef::Init***
 - Usart communication parameters
- ***uint8_t* USART_HandleTypeDef::pTxBuffPtr***
 - Pointer to Usart Tx transfer Buffer
- ***uint16_t USART_HandleTypeDef::TxXferSize***
 - Usart Tx Transfer size
- ***__IO uint16_t USART_HandleTypeDef::TxXferCount***
 - Usart Tx Transfer Counter
- ***uint8_t* USART_HandleTypeDef::pRxBuffPtr***
 - Pointer to Usart Rx transfer Buffer
- ***uint16_t USART_HandleTypeDef::RxXferSize***
 - Usart Rx Transfer size
- ***__IO uint16_t USART_HandleTypeDef::RxXferCount***
 - Usart Rx Transfer Counter
- ***uint16_t USART_HandleTypeDef::Mask***
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx***

- Usart Tx DMA Handle parameters
- **DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx**
 - Usart Rx DMA Handle parameters
- **HAL_LockTypeDef USART_HandleTypeDef::Lock**
 - Locking object
- **__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State**
 - Usart communication state
- **__IO HAL_USART_ErrorTypeDef USART_HandleTypeDef::ErrorCode**
 - USART Error code

45.2.2 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32l0xx_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: Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
- **uint32_t USART_InitTypeDef::WordLength**
 - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [USARTEx_Word_Length](#)
- **uint32_t USART_InitTypeDef::StopBits**
 - Specifies the number of stop bits transmitted. This parameter can be a value of [USART_Stop_Bits](#)
- **uint32_t USART_InitTypeDef::Parity**
 - Specifies the parity mode. This parameter can be a value of [USART_Parity](#)
- **uint32_t USART_InitTypeDef::Mode**
 - Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [USART_Mode](#)
- **uint32_t USART_InitTypeDef::CLKPolarity**
 - Specifies the steady state of the serial clock. This parameter can be a value of [USART_Clock_Polarity](#)
- **uint32_t USART_InitTypeDef::CLKPhase**
 - Specifies the clock transition on which the bit capture is made. This parameter can be a value of [USART_Clock_Phase](#)
- **uint32_t USART_InitTypeDef::CLKLastBit**

- Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of **USART_Last_Bit**

45.2.3 USART_TypeDef

USART_TypeDef is defined in the stm32l051xx.h

Data Fields

- **_IO uint32_t CR1**
- **_IO uint32_t CR2**
- **_IO uint32_t CR3**
- **_IO uint32_t BRR**
- **_IO uint16_t GTPR**
- **uint16_t RESERVED2**
- **_IO uint32_t RTOR**
- **_IO uint16_t RQR**
- **uint16_t RESERVED3**
- **_IO uint32_t ISR**
- **_IO uint32_t ICR**
- **_IO uint16_t RDR**
- **uint16_t RESERVED4**
- **_IO uint16_t TDR**
- **uint16_t RESERVED5**

Field Documentation

- **_IO uint32_t USART_TypeDef::CR1**
 - USART Control register 1, Address offset: 0x00
- **_IO uint32_t USART_TypeDef::CR2**
 - USART Control register 2, Address offset: 0x04
- **_IO uint32_t USART_TypeDef::CR3**
 - USART Control register 3, Address offset: 0x08
- **_IO uint32_t USART_TypeDef::BRR**
 - USART Baud rate register, Address offset: 0x0C
- **_IO uint16_t USART_TypeDef::GTPR**
 - USART Guard time and prescaler register, Address offset: 0x10
- **uint16_t USART_TypeDef::RESERVED2**
 - Reserved, 0x12
- **_IO uint32_t USART_TypeDef::RTOR**
 - USART Receiver Time Out register, Address offset: 0x14
- **_IO uint16_t USART_TypeDef::RQR**
 - USART Request register, Address offset: 0x18
- **uint16_t USART_TypeDef::RESERVED3**
 - Reserved, 0x1A
- **_IO uint32_t USART_TypeDef::ISR**
 - USART Interrupt and status register, Address offset: 0x1C
- **_IO uint32_t USART_TypeDef::ICR**
 - USART Interrupt flag Clear register, Address offset: 0x20

- **`_IO uint16_t USART_TypeDef::RDR`**
 - USART Receive Data register, Address offset: 0x24
- **`uint16_t USART_TypeDef::RESERVED4`**
 - Reserved, 0x26
- **`_IO uint16_t USART_TypeDef::TDR`**
 - USART Transmit Data register, Address offset: 0x28
- **`uint16_t USART_TypeDef::RESERVED5`**
 - Reserved, 0x2A

45.3 USART Firmware driver API description

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

45.3.1 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits).
 - USART polarity
 - USART phase
 - USART Last Bit
 - 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()`**

45.3.2 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the

DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL_USART_ErrorCallback() user callback will be executed when a communication error is detected.

2. Blocking mode API's are :
 - HAL_USART_Transmit() in simplex mode
 - HAL_USART_Receive() in full duplex receive only
 - HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
 - HAL_USART_Transmit_IT() in simplex mode
 - HAL_USART_Receive_IT() in full duplex receive only
 - HAL_USART_TransmitReceive_IT() in full duplex mode
 - HAL_USART_IRQHandler()
4. No-Blocking mode functions with DMA are :
 - HAL_USART_Transmit_DMA() in simplex mode
 - HAL_USART_Receive_DMA() in full duplex receive only
 - HAL_USART_TransmitReceive_DMA() in full duplex mode
 - HAL_USART_DMAPause()
 - HAL_USART_DMAResume()
 - HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
 - HAL_USART_TxCpltCallback()
 - HAL_USART_RxCpltCallback()
 - HAL_USART_TxHalfCpltCallback()
 - HAL_USART_RxHalfCpltCallback()
 - 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()***

45.3.3 Peripheral State functions

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

- HAL_USART_GetState() API can be helpful to check in run-time the state of the USART peripheral.
- HAL_USART_GetError() API can be helpful to check in run-time the Error Code of the USART peripheral.
- USART_SetConfig() API is used to set the USART communication parameters.
- USART_CheckIdleState() API ensures that TEACK and/or REACK bits are set after initialization
- ***HAL_USART_GetState()***
- ***HAL_USART_GetError()***

45.3.3.1 HAL_USART_Init

Function Name	HAL_StatusTypeDef HAL_USART_Init (<i>USART_HandleTypeDef * husart</i>)
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.2 HAL_USART_DelInit

Function Name	HAL_StatusTypeDef HAL_USART_DelInit (<i>USART_HandleTypeDef * husart</i>)
Function Description	Deinitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • HAL status

45.3.3.3 HAL_USART_MspInit

Function Name	void HAL_USART_MspInit (<i>USART_HandleTypeDef * husart</i>)
Function Description	USART MSP Init.

Parameters	<ul style="list-style-type: none"> husart : USART handle
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

45.3.3.4 HAL_USART_MspDeInit

Function Name	void HAL_USART_MspDeInit (<i>USART_HandleTypeDef</i> * husart)
Function Description	USART MSP DeInit.
Parameters	<ul style="list-style-type: none"> husart : USART handle
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

45.3.3.5 HAL_USART_Transmit

Function Name	HAL_StatusTypeDef HAL_USART_Transmit (<i>USART_HandleTypeDef</i> * husart, <i>uint8_t</i> * pTxData, <i>uint16_t</i> Size, <i>uint32_t</i> Timeout)
Function Description	Simplex Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> husart : USART handle pTxData : Pointer to data buffer Size : Amount of data to be sent Timeout : Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> None.

45.3.3.6 HAL_USART_Receive

Function Name	HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> • husart : USART handle • pRxData : pointer to data buffer • Size : amount of data to be received • Timeout : : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.7 HAL_USART_TransmitReceive

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart : USART handle • pTxData : pointer to TX data buffer • pRxData : pointer to RX data buffer • Size : amount of data to be sent (same amount to be received) • Timeout : : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.8 HAL_USART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • husart : USART handle

	<ul style="list-style-type: none"> • pTxData : Pointer to data buffer • Size : Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.9 HAL_USART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> • husart : usart handle • pRxData : pointer to data buffer • Size : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.10 HAL_USART_TransmitReceive_IT

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • husart : USART handle • pTxData : pointer to TX data buffer • pRxData : pointer to RX data buffer • Size : amount of data to be sent (same amount to be received)
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.11 HAL_USART_Transmit_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)</code>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • husart : USART handle • pTxData : pointer to data buffer • Size : amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

45.3.3.12 HAL_USART_Receive_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)</code>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • husart : USART handle • pRxData : pointer to data buffer • Size : amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position) • The USART DMA transmit stream must be configured in order to generate the clock for the slave.

45.3.3.13 HAL_USART_TransmitReceive_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t *)</code>
---------------	---

pRxData, uint16_t Size)

Function Description	Full-Duplex Transmit Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> • husart : usart handle • pTxData : pointer to TX data buffer • pRxData : pointer to RX data buffer • Size : amount of data to be received/sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

45.3.3.14 HAL_USART_DMAPause

Function Name	HAL_StatusTypeDef HAL_USART_DMAPause (<i>USART_HandleTypeDef</i> * husart)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.

45.3.3.15 HAL_USART_DMAResume

Function Name	HAL_StatusTypeDef HAL_USART_DMAResume (<i>USART_HandleTypeDef</i> * husart)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.

45.3.3.16 HAL_USART_DMAMStop

Function Name	HAL_StatusTypeDef HAL_USART_DMAMStop (<i>USART_HandleTypeDef</i> * husart)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

45.3.3.17 HAL_USART_IRQHandler

Function Name	void HAL_USART_IRQHandler (<i>USART_HandleTypeDef</i> * husart)
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

45.3.3.18 HAL_USART_TxCpltCallback

Function Name	void HAL_USART_TxCpltCallback (<i>USART_HandleTypeDef</i> * husart)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

45.3.3.19 HAL_USART_TxHalfCpltCallback

Function Name	void HAL_USART_TxHalfCpltCallback (<i>USART_HandleTypeDef</i> * husart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart : USART handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

45.3.3.20 HAL_USART_RxCpltCallback

Function Name	void HAL_USART_RxCpltCallback (<i>USART_HandleTypeDef</i> * husart)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart : USART handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

45.3.3.21 HAL_USART_RxHalfCpltCallback

Function Name	void HAL_USART_RxHalfCpltCallback (<i>USART_HandleTypeDef</i> * husart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart : USART handle
Return values	<ul style="list-style-type: none">• None.
Notes	<ul style="list-style-type: none">• None.

45.3.3.22 HAL_USART_TxRxCpltCallback

Function Name	void HAL_USART_TxRxCpltCallback (<i>USART_HandleTypeDef</i> * husart)
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

45.3.3.23 HAL_USART_ErrorCallback

Function Name	void HAL_USART_ErrorCallback (<i>USART_HandleTypeDef</i> * husart)
Function Description	USART error callbacks.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

45.3.3.24 HAL_USART_GetState

Function Name	HAL_USART_StateTypeDef HAL_USART_GetState (<i>USART_HandleTypeDef</i> * husart)
Function Description	Returns the USART state.
Parameters	<ul style="list-style-type: none"> • husart : USART handle
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • None.

45.3.3.25 HAL_USART_GetError

Function Name	<code>uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)</code>
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> husart : : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	<ul style="list-style-type: none"> USART Error Code
Notes	<ul style="list-style-type: none"> None.

45.4 USART Firmware driver defines

45.4.1 USART

USART

USART_Clock

- #define: `USART_CLOCK_DISABLED ((uint32_t)0x0000)`
- #define: `USART_CLOCK_ENABLED ((uint32_t)USART_CR2_CLKEN)`

USART_Clock_Phase

- #define: `USART_PHASE_1EDGE ((uint32_t)0x0000)`
- #define: `USART_PHASE_2EDGE ((uint32_t)USART_CR2_CPHA)`

USART_Clock_Polarity

- #define: `USART_POLARITY_LOW ((uint32_t)0x0000)`
- #define: `USART_POLARITY_HIGH ((uint32_t)USART_CR2_CPOL)`

USART_Flags

- #define: ***USART_FLAG_RXACK*** ((*uint32_t*)0x00400000)
- #define: ***USART_FLAG_TEACK*** ((*uint32_t*)0x00200000)
- #define: ***USART_FLAG_BUSY*** ((*uint32_t*)0x00010000)
- #define: ***USART_FLAG_CTS*** ((*uint32_t*)0x00000400)
- #define: ***USART_FLAG_CTSIF*** ((*uint32_t*)0x00000200)
- #define: ***USART_FLAG_LBDF*** ((*uint32_t*)0x00000100)
- #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_Interruption_Mask

- #define: **USART_IT_MASK** ((*uint16_t*)0x001F)

USART Interrupt definition

- #define: **USART_IT_PE** ((*uint16_t*)0x0028)
- #define: **USART_IT_TXE** ((*uint16_t*)0x0727)
- #define: **USART_IT_TC** ((*uint16_t*)0x0626)
- #define: **USART_IT_RXNE** ((*uint16_t*)0x0525)
- #define: **USART_IT_IDLE** ((*uint16_t*)0x0424)
- #define: **USART_IT_ERR** ((*uint16_t*)0x0060)
- #define: **USART_IT_ORE** ((*uint16_t*)0x0300)
- #define: **USART_IT_NE** ((*uint16_t*)0x0200)

- #define: **USART_IT_FE** ((*uint16_t*)0x0100)

USART_IT_CLEAR_Flags

- #define: **USART_CLEAR_PEF USART_ICR_PECF**
Parity Error Clear Flag
- #define: **USART_CLEAR_FEF USART_ICR_FECF**
Framing Error Clear Flag
- #define: **USART_CLEAR_NEF USART_ICR_NCF**
Noise detected Clear Flag
- #define: **USART_CLEAR_OREF USART_ICR_ORECF**
OverRun Error Clear Flag
- #define: **USART_CLEAR_IDLEF USART_ICR_IDLECF**
IDLE line detected Clear Flag
- #define: **USART_CLEAR_TCF USART_ICR_TCCF**
Transmission Complete Clear Flag
- #define: **USART_CLEAR_CTSF USART_ICR_CTSCF**
CTS Interrupt Clear Flag

USART_Last_Bit

- #define: **USART_LASTBIT_DISABLE** ((*uint32_t*)0x0000)
- #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_RX** ((*uint32_t*)(**USART_CR1_TE | USART_CR1_RE**))

USART_Parity

- #define: **USART_PARITY_NONE** ((*uint32_t*)0x0000)
- #define: **USART_PARITY_EVEN** ((*uint32_t*)**USART_CR1_PCE**)
- #define: **USART_PARITY_ODD** ((*uint32_t*)(**USART_CR1_PCE | USART_CR1_PS**))

USART_Request_Parameters

- #define: **USART_RXDATA_FLUSH_REQUEST** ((*uint32_t*)**USART_RQR_RXFRQ**)
Receive Data flush Request
- #define: **USART_TXDATA_FLUSH_REQUEST** ((*uint32_t*)**USART_RQR_TXFRQ**)
Transmit data flush Request

USART_Stop_Bits

- #define: **USART_STOPBITS_1** ((*uint32_t*)0x0000)
- #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**)))

46 HAL USART Extension Driver

46.1 USARTEx Firmware driver defines

46.1.1 USARTEx

USARTEx

USARTEx_Word_Length

- #define: ***USART_WORDLENGTH_7B*** ((*uint32_t*)***USART_CR1_M_1***)

- #define: ***USART_WORDLENGTH_8B*** ((*uint32_t*)0x00000000)

- #define: ***USART_WORDLENGTH_9B*** ((*uint32_t*)***USART_CR1_M_0***)

47 HAL WWDG Generic Driver

47.1 WWDG Firmware driver introduction

47.2 WWDG Firmware driver registers structures

47.2.1 WWDG_HandleTypeDefDef

WWDG_HandleTypeDefDef is defined in the `stm32l0xx_hal_wwdg.h`

Data Fields

- *WWDG_TypeDef * Instance*
- *WWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_WWDG_StateTypeDef State*

Field Documentation

- *WWDG_TypeDef* WWDG_HandleTypeDefDef::Instance*
 - Register base address
- *WWDG_InitTypeDef WWDG_HandleTypeDefDef::Init*
 - WWDG required parameters
- *HAL_LockTypeDef WWDG_HandleTypeDefDef::Lock*
 - WWDG locking object
- *__IO HAL_WWDG_StateTypeDef WWDG_HandleTypeDefDef::State*
 - WWDG communication state

47.2.2 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the `stm32l0xx_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

47.2.3 WWDG_TypeDef

WWDG_TypeDef is defined in the stm32l051xx.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

47.3 WWDG Firmware driver API description

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

47.3.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDRST flag in RCC_CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG counter clock = PCLK1 / Prescaler WWDG timeout = (WWDG counter clock)
* (counter value)
- Min-max timeout value @32 MHz(PCLK1): ~128.0 us / ~65.54 ms

47.3.2 How to use this driver

- Enable WWDG APB1 clock using `__WWDG_CLK_ENABLE()`.
- Set the WWDG prescaler, refresh window and counter value using `HAL_WWDG_Init()` function.
- Start the WWDG using `HAL_WWDG_Start()` function. When the WWDG is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWDG using `HAL_WWDG_Start_IT()`. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using `HAL_WWDG_Refresh()` function. This operation must occur only when the counter is lower than the refresh window value already programmed.

WWDG HAL driver macros list

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

- `__HAL_WWDG_ENABLE`: Enable the WWDG peripheral
- `__HAL_WWDG_GET_FLAG`: Get the selected WWDG's flag status
- `__HAL_WWDG_CLEAR_FLAG`: Clear the WWDG's pending flags
- `__HAL_WWDG_ENABLE_IT`: Enables the WWDG early wakeup interrupt

47.3.3 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()`

47.3.4 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()`

47.3.5 Peripheral State functions

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

- ***HAL_WWDG_GetState()***

47.3.5.1 HAL_WWDG_Init

Function Name	HAL_StatusTypeDef HAL_WWDG_Init (<i>WWDG_HandleTypeDef * hwdg</i>)
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

47.3.5.2 HAL_WWDG_DeInit

Function Name	HAL_StatusTypeDef HAL_WWDG_DeInit (<i>WWDG_HandleTypeDef * hwdg</i>)
Function Description	Deinitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

47.3.5.3 HAL_WWDG_MspInit

Function Name	void HAL_WWDG_MspInit (<i>WWDG_HandleTypeDef * hwdg</i>)
---------------	---

Function Description	Initializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

47.3.5.4 HAL_WWDG_MspDeInit

Function Name	void HAL_WWDG_MspDeInit (<i>WWDG_HandleTypeDef</i> * hwdg)
Function Description	Deinitializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • None.
Notes	<ul style="list-style-type: none"> • None.

47.3.5.5 HAL_WWDG_Start

Function Name	HAL_StatusTypeDef HAL_WWDG_Start (<i>WWDG_HandleTypeDef</i> * hwdg)
Function Description	Starts the WWDG.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

47.3.5.6 HAL_WWDG_Start_IT

Function Name	HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDef * hwdg)
Function Description	Starts the WWDG with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

47.3.5.7 HAL_WWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwdg, uint32_t Counter)
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • None.

47.3.5.8 HAL_WWDG_IRQHandler

Function Name	void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwdg)
Function Description	Handles WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> • hwdg : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • None.

Notes	<ul style="list-style-type: none"> 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 <code>__HAL_WWDG_ENABLE_IT()</code> 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.
-------	---

47.3.5.9 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (<i>WWDG_HandleTypeDefDef</i> * hwdg)
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> hwdg : pointer to a <i>WWDG_HandleTypeDefDef</i> structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None.
Notes	<ul style="list-style-type: none"> None.

47.3.5.10 HAL_WWDG_GetState

Function Name	HAL_WWDG_StateTypeDef HAL_WWDG_GetState (<i>WWDG_HandleTypeDefDef</i> * hwdg)
Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> hwdg : pointer to a <i>WWDG_HandleTypeDefDef</i> structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> HAL state
Notes	<ul style="list-style-type: none"> None.

47.4 WWDG Firmware driver defines

47.4.1 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

48 Revision history

Table 16: Document revision history

Date	Revision	Changes
03-Jun-2014	1	Initial release.

Please Read Carefully

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at anytime, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER'S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR "AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL" INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2014 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com